Incremental Kernel Principal Component Analysis

Tat-Jun Chin and David Suter, *Senior Member, IEEE*

***Abstract*—The kernel principal component analysis (KPCA) has been applied in numerous image-related machine learning applications and it has exhibited superior performance over previous approaches, such as PCA. However, the standard implementation of KPCA scales badly with the problem size, making computations for large problems infeasible. Also, the "batch" nature of the standard KPCA computation method does not allow for applications that require online processing. This has somewhat restricted the domains in which KPCA can potentially be applied. This paper introduces an incremental computation algorithm for KPCA to address these two problems. The basis of the proposed solution lies in computing incremental linear PCA in the kernel induced feature space, and constructing reduced-set expansions to maintain constant update speed and memory usage. We also provide experimental results which demonstrate the effectiveness of the approach.**

***Index Terms*—Enabling online processing, incremental kernel principal component analysis (KPCA), reduced-set expansions, reducing time complexity.**

I. Introduction

CONVENTIONAL linear subspace methods such as principal component analysis (PCA) and Fisher's discriminant analysis (FDA) can only produce linear subspace feature extractors. These are unsuitable for highly complex and nonlinear data distributions. In contrast, kernel subspace methods such as kernel PCA (KPCA) and kernel FDA (KFDA) can capture higher-order statistics present in a dataset, thus producing nonlinear subspaces for better feature extraction. In principle, the kernel methods function by nonlinearly mapping a set of training data to a higher dimensional feature space where conventional linear subspace methods are performed, with the resulting subspaces being "nonlinear" with regards to the original input space. In practice, the mapping is performed implicitly via the "kernel trick," where an appropriately chosen kernel function is used to evaluate dot products of mapped input space vectors without having to explicitly carry out the mapping. In this paper our focus is on KPCA which has been used in a wide range of applications such as face recognition [1], single frame super-resolution [2], image denoising [3], and acquisition of multiple view feature descriptors [4]. In experiments where comparisons have been made, KPCA almost always outperforms standard PCA.

In order to obtain accurate nonlinear principal components to describe a complex data distribution, a large number of training samples are required, especially for data embedded in a high-dimensional space. This presents a difficulty for KPCA since it requires that the whole dataset be stored and manipulated at once. The standard KPCA computation method [5] via eigendecompositions involves a time complexity of $\mathcal{O}(n^3)$, with n being the number of training vectors, thus evaluating the KPCA is prohibitively expensive for large datasets. Second, the resulting kernel principal components have to be defined implicitly by linear expansions of the training data; thus, all data must be saved after training. For massive datasets, this translates into high costs for storage resources and computational load during run-time application of kernel principal components. Furthermore, for applications that require online data processing, KPCA is completely impracticable since it is computable in a batch manner only, i.e., the new data is appended to the dataset and the KPCA process is restarted. To overcome these limitations, we propose an incremental KPCA computation method. The essence of our solution lies in performing incremental PCA in the kernel induced feature space [6], [7] and using reduced-set (RS) expansions [6] to maintain nonincreasing memory usage and update duration. The objectives and contributions of our work can be summarized as:

1) reducing the computational complexity of KPCA to $\mathcal{O}(n)$ to facilitate processing of large datasets;
2) endowing KPCA with the capability of updating previous computations with novel data.

We also provide a theoretical analysis of the time complexity of the proposed method and experimental results to support the proposed algorithm.

The rest of the paper is organized as follows. Section II surveys related work to put this paper in context, and Section III describes briefly the KPCA method. Section IV elucidates the proposed method, covering also RS compressions, KPCA basis re-orthogonalization and analysis of complexity. Section V details the experimental results. We derive the conclusion in Section VI.

II. Related Work

All kernel methods involve computing and processing the *kernel matrix* which behaves like an *information bottleneck* [7] from which the kernel methods infer the characteristics of the dataset after it is nonlinearly mapped to a higher dimensional

Manuscript received May 14, 2006; revised February 21, 2007. This work was conducted when T.-J. Chin was with the Department of Electrical and Computer Systems Engineering, Monash University. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Benoit Macq.

T.-J. Chin is with the Institute for Infocomm Research, Singapore 119613 (e-mail: tjchin@i2r.a-star.edu.sg).

D. Suter is with the Department of Electrical and Computer Systems Engineering, Monash University, 3800 Victoria, Australia (e-mail: d.suter@eng.monash.edu.au).

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

Digital Object Identifier 10.1109/TIP.2007.896668

feature space. For KPCA, an eigendecomposition or, equivalently, a singular value decomposition (SVD) is invoked on the kernel matrix. For large datasets, direct factorizations on the kernel matrix become infeasible (see Section IV). An obvious solution is to factorize the kernel matrix incrementally, i.e., factorize the matrix by using a small number of columns at a time. This is practiced in [8], where the process is interpreted from the perspective of the empirical kernel map (EKM) [9]. Put simply, this mapping views each vector to have been transformed to a corresponding column in the kernel matrix. This exempts us from storing and processing the whole kernel matrix at once. Nonetheless, an obvious flaw of the method is that the EKM is dataset specific. In other words, the column length increases with the number of data. Given a novel datum, a new EKM has to be defined (i.e., a new row *and* column are added to the kernel matrix) and the factorization must be re-evaluated. This cannot be utilized in applications such as [10] and [11] which require *true* incremental KPCA.

Previously, we proposed an incremental kernel SVD (KSVD) algorithm which is based on kernelizing incremental linear SVD [12]. This does not imply incremental KPCA is solved, since incremental KSVD does not require adaptive centering of the incremental data and the appropriate adjustment of the factorized subspace bases. Note that KPCA and KSVD return vastly different results on the same dataset if it is not centered. The work in this paper takes updating the mean into account. In [12], reduced set expansions are constructed to compress the KSVD basis so as to achieve constant incremental update speed. We improve upon this by proposing a better compression strategy. In addition, a kernel subspace re-orthogonalization scheme is inserted in the proposed algorithm. These steps can dramatically increase the approximation accuracy of incremental KPCA and retrospectively for incremental KSVD.

The kernel Hebbian algorithm (KHA) was proposed as an iterative KPCA algorithm in [2]. Essentially, the method involves kernelizing the generalized Hebbian algorithm (GHA) which is an online computation procedure for linear PCA. Similar in operational characteristics to single-layer feedforward neural networks, the KHA outputs converge towards the desired kernel principal components by *iterating* the algorithm using the intended training data over *multiple passes*. While this can potentially lower the time complexity of computing KPCA, it is unclear how novel data can be incorporated to update the KHA outputs. Most likely, it is added to the previous training set and the iterations are restarted. Here, we seek an *incremental* computation algorithm for KPCA such that it is unnecessary to consider all available data more than once, and novel data can be used to update a previous computation easily.

Another important development is the greedy KPCA [13], [14] which essentially works by *filtering* or *sampling* the original training set for a lesser but representative subset of vectors which span approximately the same subspace as the subspace in the kernel induced feature space spanned by the training set. The training set is then projected onto the span of the lesser subset, where PCA is carried out. Since the dimension of the span of the lesser subset is smaller than the number of training vectors, computational effort is reduced. However, one drawback is the prior filtering of the training data which could be computationally expensive by itself. Other sampling-based methods exist [15]–[18]. However, our solution not only tackles the time complexity issue, it has the capability of *updating* a previous computation with data unavailable initially in a manner that maintains nonincreasing memory usage and update duration. This is crucial for applications that require online processing of KPCA.

In [4], KPCA was used to obtain feature descriptors from multiple images for application in mobile robot navigation and localization. After processing the training images, RS expansions are constructed to compress the kernel principal component representations to reduce computational load during runtime application. However, no method was proposed to update the kernel principal components as more features become available. Hence, this is not incremental KPCA.

III. Kernel PCA Feature Extraction

We first establish notation. Given a matrix $\mathbf{M}$, the symbol $\mathbf{M}_{a:b,c:d}$ defines the submatrix that contains the elements of $\mathbf{M}$ within the intersection of the ath row till the b-row and the cth column till the dth column. If the row or column specifiers are omitted, take all available rows or columns, e.g., for $\mathbf{M}_{:,c:d}$ take all rows but only the cth to dth columns. $\mathbf{I}_n$ indicates an identity matrix of size $n \times n$, $\mathbf{1}_{r,c}$ represents a matrix of ones of size $r \times c$, while $\mathbf{0}_{n,c}$ symbolizes an $n \times c$ zero matrix.

We begin by obtaining a data matrix $\mathbf{a} = [\mathbf{x}_1 \cdots \mathbf{x}_n] \in \mathbb{R}^{m \times n}$, with $\mathbf{x}_i \in \mathbb{R}^m$ being the ith input vector. To perform KPCA, in principle we nonlinearly map $\mathbf{a}$ to a higher dimensional space $\mathcal{F}$ using the function $\phi : \mathbb{R}^m \longrightarrow \mathcal{F}$ and perform PCA in $\mathcal{F}$. Using ϕ, we transform $\mathbf{a}$ into $\mathbf{A} = [\phi(\mathbf{x}_1) \cdots \phi(\mathbf{x}_n)]$. The map ϕ is induced by a kernel function $k(\cdot, \cdot)$ that allows us to evaluate inner products in $\mathcal{F}$

$$k(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z}), \text{ with } \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^m. \tag{1}$$

$k(\cdot, \cdot)$ must be an appropriately chosen Mercer kernel so that ϕ belongs to a function space that has the structure of a so-called reproducing kernel Hilbert space (RKHS) [6], [7]. Note that $\mathbf{A}$ could be embedded in a very high-dimensional space, and we would not want to explicitly evaluate $\phi(\mathbf{a})$.

We center $\mathbf{A}$ by subtracting the mean $\boldsymbol{\mu}_{\mathbf{A}}$ from $\mathbf{A}$. The mean and mean-adjusted data are, respectively

$$\mu_{\mathbf{A}} = \mathbf{A}\left(\frac{1}{n}\mathbf{1}_{n,1}\right) = \mathbf{A}\boldsymbol{\nu} \tag{2}$$

$$\hat{\mathbf{A}} = \mathbf{A}(\mathbf{I}_n - \nu\mathbf{1}_{1,n}) = \mathbf{A}\boldsymbol{\nu}' \tag{3}$$

where $\boldsymbol{\nu} := ((1/n)\mathbf{1}_{n,1})$ and $\boldsymbol{\nu}' := (\mathbf{I}_n - \boldsymbol{\nu}\mathbf{1}_{1,n})$. Consider the matrix

$$\mathbf{M} = \hat{\mathbf{A}}^T\hat{\mathbf{A}} = (\boldsymbol{\nu}')^T\mathbf{A}^T\mathbf{A}\boldsymbol{\nu}' \tag{4}$$

and its eigenvalue decomposition $\mathbf{M} = \mathbf{Q}\boldsymbol{\Delta}\mathbf{Q}^T$. By using $k(\cdot, \cdot)$, $\mathbf{A}^T\mathbf{A}$ can be evaluated without having to perform the mapping ϕ since $\mathbf{A}^T\mathbf{A}$ contains only dot products between the $\phi(\mathbf{x}_i)$s. Matrix $\mathbf{M}$ is the kernel matrix for KPCA. Via kernel SVD [7], the rank-r singular value factorization of $\hat{\mathbf{A}}$ is

$$\hat{\mathbf{A}}^r = \left[\hat{\mathbf{A}}\mathbf{Q}^r(\boldsymbol{\Delta}^r)^{-\frac{1}{2}}\right]\left[(\boldsymbol{\Delta}^r)^{\frac{1}{2}}\right]\left[(\mathbf{Q}^r)^T\right] \equiv \mathbf{U}^r\boldsymbol{\Sigma}^r(\mathbf{V}^r)^T \tag{5}$$

where $\mathbf{Q}^r = \mathbf{Q}_{:,1:r}$ and $\mathbf{\Delta}^r = \mathbf{\Delta}_{1:r,1:r}$. Matrix $\mathbf{M}$ is positive semi-definite if it is constructed using a valid Mercer kernel [6], [7].

The columns of $\mathbf{U}^r$ are the r most significant principal components of $\mathbf{A}$ (i.e., the kernel principal components of $\mathbf{a}$) ranked according to $diag(\sqrt{\mathbf{\Delta}^r})$. Observe that $\mathbf{U}^r$ are defined implicitly by linear expansions of mapped input data

$$\mathbf{U}^r = \mathbf{A}\boldsymbol{\nu}'\mathbf{Q}^r(\mathbf{\Delta}^r)^{-\frac{1}{2}} = \mathbf{A}\boldsymbol{\alpha} \tag{6}$$

where $\boldsymbol{\alpha} := \boldsymbol{\nu}'\mathbf{Q}^r(\mathbf{\Delta}^r)^{-(1/2)}$ contains the expansion coefficients. Let $\mathbf{A}^r$ be the reconstruction of $\mathbf{A}$ using the first-r kernel principal components, i.e., $\mathbf{A}^r = \hat{\mathbf{A}}^r + \boldsymbol{\mu}_{\mathbf{A}}$. It is crucial that the data distribution in $\mathcal{F}$ lies near an r-dimensional subspace (where $r \ll n$) for $\mathbf{A}^r$ to approximate $\mathbf{A}$ closely. This is achievable by choosing the *appropriate* kernel function.

The orthogonal projection coefficients of data onto the kernel principal components in the feature space $\mathcal{F}$ can be considered to be extracted nonlinear features. Given data $\mathbf{b}$, we center and project it onto the kernel principal components $\mathbf{U}^r$ by

$$\boldsymbol{\beta} = (\mathbf{U}^r)^T(\mathbf{B} - \boldsymbol{\mu}_{\mathbf{A}}) = \boldsymbol{\alpha}^T\mathbf{A}^T[\mathbf{A} \quad \mathbf{B}]\begin{bmatrix}-\boldsymbol{\nu}\\ 1\end{bmatrix} \tag{7}$$

where $\mathbf{B} = \phi(\mathbf{b})$ and $\boldsymbol{\beta}$ contains the nonlinear features extracted from $\mathbf{b}$. The matrix $\mathbf{A}^T[\mathbf{A}\ \mathbf{B}]$ can be evaluated using the kernel function since it contains only dot products between feature space vectors.

IV. Computing Kernel PCA Incrementally

For KPCA, as shown in Section III, the kernel matrix $\mathbf{M}$ has to be wholly available before eigendecompositions can be carried out. Since the size of $\mathbf{M}$ scales with n^2, where n is the number of training vectors, a large memory is required for substantial datasets. A more crippling problem is the eigendecomposition of $\mathbf{M}$ which involves a time complexity of $\mathcal{O}(n^3)$. This can severely handicap the computation of KPCA on large datasets. In applications that require online processing, the arrival of a new vector will require the addition of a new row and column for $\mathbf{M}$. Furthermore, eigendecompositions have to be constantly re-evaluated for an ever-growing kernel matrix in order to obtain updated kernel subspaces. Hence, batch KPCA is infeasible for such applications.

A. Proposed Solution

Part of our proposed solution for incremental KPCA involves kernelizing an exact incremental PCA [10] algorithm. Note that although the following derivations make references to feature space vectors, their explicit existence is never required and the mapping ϕ is always avoided. To begin, we proceed from Section III where KPCA has been performed on training data $\mathbf{a} \in \mathbb{R}^{m\times n}$, so the same definitions of symbols apply here. To update the KPCA given new data, we should not just append the new data to $\mathbf{a}$ and reevaluate KPCA since this would correspond to batch computation.

An alternative approach needs to be derived. Given new data $\mathbf{c} \in \mathbb{R}^{m\times i}$, the PCA of the overall data $\mathbf{D} = [\mathbf{A}^r\mathbf{C}]$ is sought, where $\mathbf{C} = \phi(\mathbf{c})$. The mean of $\mathbf{C}$ can be updated as

$$\boldsymbol{\mu}_{\mathbf{C}} = \mathbf{C}\boldsymbol{\omega}, \text{ with } \boldsymbol{\omega} := \frac{1}{i}\mathbf{1}_{i,1}. \tag{8}$$

Trivially, the mean of the overall data $\mathbf{D}$ can be updated as

$$\begin{aligned}\boldsymbol{\mu}_{\mathbf{D}} &= \frac{n}{n+i}\mathbf{A}\boldsymbol{\nu} + \frac{i}{n+i}\mathbf{C}\boldsymbol{\omega} = \bar{\mathbf{A}}\bar{\boldsymbol{\nu}},\\ \text{with } \bar{\mathbf{A}} &:= [\mathbf{A} \quad \mathbf{C}] \text{ and } \bar{\boldsymbol{\nu}} := \frac{1}{n+i}\begin{bmatrix}n\boldsymbol{\nu}\\ i\boldsymbol{\omega}\end{bmatrix}.\end{aligned} \tag{9}$$

Following (3), $\mathbf{C}$ is centered with regards to $\boldsymbol{\mu}_{\mathbf{C}}$ as

$$\hat{\mathbf{C}} = \mathbf{C}(\mathbf{I}_i - \boldsymbol{\omega}\mathbf{1}_{1,i}) = \mathbf{C}\boldsymbol{\omega}' \tag{10}$$

with $\boldsymbol{\omega}' := (\mathbf{I}_i - \boldsymbol{\omega}\mathbf{1}_{1,i})$. At this stage, we use these intermediate results to construct matrix $\tilde{\mathbf{E}}$ which is defined as

$$\begin{aligned}\tilde{\mathbf{E}} &= \begin{bmatrix}\hat{\mathbf{C}} & \sqrt{\frac{ni}{n+i}}(\boldsymbol{\mu}_{\mathbf{A}} - \boldsymbol{\mu}_{\mathbf{C}})\end{bmatrix}\\ &= [\mathbf{A} \quad \mathbf{C}]\begin{bmatrix}\begin{bmatrix}\mathbf{0}_{n,i}\\ \omega'\end{bmatrix} & \sqrt{\frac{ni}{n+i}}\begin{bmatrix}\boldsymbol{\nu}\\ -\boldsymbol{\omega}\end{bmatrix}\end{bmatrix} = \bar{\mathbf{A}}\boldsymbol{\gamma}\end{aligned}$$

with $\boldsymbol{\gamma} := \left[\begin{bmatrix}\mathbf{0}_{n,i}\\ \boldsymbol{\omega}'\end{bmatrix} \sqrt{ni/(n+i)}\begin{bmatrix}\boldsymbol{\nu}\\ -\boldsymbol{\omega}\end{bmatrix}\right]$. The significance of $\tilde{\mathbf{E}}$ will be revealed shortly. Both $\boldsymbol{\mu}_{\mathbf{D}}$ and $\tilde{\mathbf{E}}$ are linear expansions of old and new data, i.e., $\mathbf{A} = \phi(\mathbf{a})$ and $\mathbf{C} = \phi(\mathbf{c})$.

To perform PCA on $\mathbf{D}$, we can center $\mathbf{D}$ with regards to $\boldsymbol{\mu}_{\mathbf{D}}$ and invoke the SVD. However, this would be a batch PCA since both $\mathbf{A}^r$ and $\mathbf{C}$ have to be utilized. Since it is algebraically proven [10] that the scaled covariance matrix of $\mathbf{D}$ is

$$S_{\mathbf{D}} = \hat{\mathbf{D}}\hat{\mathbf{D}}^T = [\hat{\mathbf{A}}^r \quad \tilde{\mathbf{E}}][\hat{\mathbf{A}}^r \quad \tilde{\mathbf{E}}]^T \tag{11}$$

where $\hat{\mathbf{D}} = \mathbf{D} - \boldsymbol{\mu}_{\mathbf{D}}$, to perform PCA on $\mathbf{D}$, it is sufficient to carry out an SVD on $[\hat{\mathbf{A}}^r\tilde{\mathbf{E}}]$. Hence, we can apply incremental SVD [19] to update $\mathbf{U}^r$ using $\tilde{\mathbf{E}}$ as new data to avoid accessing $\hat{\mathbf{A}}^r$. Given the previous factorization $\hat{\mathbf{A}}^r = \mathbf{U}^r\mathbf{\Sigma}^r(\mathbf{V}^r)^T$ from (5), decompose $[\hat{\mathbf{A}}^r\tilde{\mathbf{E}}]$ as

$$[\mathbf{U}^r \quad \mathbf{J}]\begin{bmatrix}\mathbf{\Sigma}^r & \mathbf{L}\\ \mathbf{0}_{i+1,r} & \mathbf{K}\end{bmatrix}\begin{bmatrix}\mathbf{V}^r & \mathbf{0}_{n,i+1}\\ \mathbf{0}_{i+1,r} & \mathbf{I}_{i+1}\end{bmatrix}^T. \tag{12}$$

The other components that make up (12) are defined as

$$\mathbf{L} = (\mathbf{U}^r)^T\tilde{\mathbf{E}} \tag{13}$$

$$\mathbf{H} = \tilde{\mathbf{E}} - \mathbf{U}^r\mathbf{L} \tag{14}$$

$$\mathbf{J}\mathbf{K} \xleftarrow{\text{QR}} \mathbf{H}. \tag{15}$$

$\mathbf{L}$ contains the projection coefficients of $\tilde{\mathbf{E}}$ onto the subspace $span(\mathbf{U}^r)$, while $\mathbf{H}$ represents new information contained in $\tilde{\mathbf{E}}$ which are normal to the previous subspace. $\mathbf{J}$ is an orthogonal basis for the subspace that is spanned by the new information, while $\mathbf{K}$ is the projection of $\mathbf{H}$ onto $span(\mathbf{J})$.

Let $\mathbf{E}$, $\mathbf{F}$ and $\mathbf{G}^T$ be respectively defined as the left, middle, and right matrix of the right-hand side of (12). We can diagonalize $\mathbf{F}$ by invoking the SVD, i.e., $\mathbf{F} = \mathbf{U}'\mathbf{\Sigma}'(\mathbf{V}')^T$, and substitute it into (12), yielding the updated SVD

$$[\hat{\mathbf{A}}^r \quad \tilde{\mathbf{E}}] = \mathbf{U}''\mathbf{\Sigma}'(\mathbf{V}'')^T \tag{16}$$

with $\mathbf{U}'' = \mathbf{E}\mathbf{U}'$ and $\mathbf{V}'' = \mathbf{G}\mathbf{V}'$. If we wish to always retain the r most significant principal components only, $\mathbf{U}^r$, $\mathbf{\Sigma}^r$ and $\mathbf{V}^r$ are then revised as

$$\mathbf{U}^r \longleftarrow \mathbf{U}''_{:,1:r} \tag{17}$$

$$\mathbf{\Sigma}^r \longleftarrow \mathbf{\Sigma}'_{1:r,1:r} \tag{18}$$

$$\mathbf{V}^r \longleftarrow \mathbf{V}''_{:,1:r}. \tag{19}$$

The updated r kernel principal components of $[\mathbf{a}\,\mathbf{c}]$ is the revised $\mathbf{U}^r$. Note that updating $\mathbf{U}^r$ is achieved without involving $\hat{\mathbf{A}}^r$ and is dependent only on $\tilde{\mathbf{E}}$ and $\mathbf{\Sigma}^r$. $\mathbf{V}^r$ can be discarded for applications that do not make use of it.

Via the kernel trick, the procedure above is rendered feasible without explicitly evaluating ϕ. We can see that the key to update $\mathbf{U}^r$ is to store $\mathbf{\Sigma}^r$ and compute $\mathbf{J}$, $\mathbf{K}$ and $\mathbf{L}$ for new data $\mathbf{C}$. Trivially, we can compute matrix $\mathbf{L}$ as

$$\mathbf{L} = (\mathbf{U}^r)^T\tilde{\mathbf{E}} = \boldsymbol{\alpha}^T\mathbf{A}^T\bar{\mathbf{A}}\boldsymbol{\gamma} \tag{20}$$

which is realizable by using the kernel function for inner products between the columns of $\mathbf{A}$ and $\bar{\mathbf{A}}$. We can define matrix $\mathbf{H}$ subsequently as

$$\mathbf{H} = \tilde{\mathbf{E}} - \mathbf{U}^r\mathbf{L} \tag{21}$$

$$= [\mathbf{A} \quad \mathbf{C}]\begin{bmatrix}\boldsymbol{\gamma}_{1:n,:} - \boldsymbol{\alpha}\mathbf{L} \\ \boldsymbol{\gamma}_{(n+1):(n+i),:}\end{bmatrix} = \bar{\mathbf{A}}\boldsymbol{\beta} \tag{22}$$

where $\boldsymbol{\beta} := \begin{bmatrix}\boldsymbol{\gamma}_{1:n,:} - \boldsymbol{\alpha}\mathbf{L} \\ \boldsymbol{\gamma}_{(n+1):(n+i),:}\end{bmatrix}$. Again, $\mathbf{H}$ is expressible as linear combinations of mapped input data with $\boldsymbol{\beta}$ containing the expansion coefficients.

Instead of orthogonalizing $\mathbf{H}$ via the QR decomposition, we derive an equivalent orthonormal basis for $\mathbf{H}$ by performing a KSVD on $\mathbf{H}$ and retain all left singular vectors to form $\mathbf{J}$. We compute the kernel matrix for $\mathbf{H}$ as

$$\mathbf{M}_\mathbf{H} = \boldsymbol{\beta}^T\bar{\mathbf{A}}^T\bar{\mathbf{A}}\boldsymbol{\beta} = \boldsymbol{\beta}^T\bar{\mathbf{M}}\boldsymbol{\beta}. \tag{23}$$

$\bar{\mathbf{M}}$ is computable using the kernel function on vectors of $\mathbf{A}$ and $\mathbf{C}$. Hence, $\bar{\mathbf{M}}$ is positive semi-definite, and this ensures the positive semi-definiteness of $\mathbf{M}_\mathbf{H}$. In addition, $rank(\mathbf{M}_\mathbf{H}) = rank(\mathbf{H})$. Let the eigendecomposition of $\mathbf{M}_\mathbf{H} = \mathbf{Q}_\mathbf{H}\mathbf{\Delta}_\mathbf{H}\mathbf{Q}_\mathbf{H}^T$. Following (5), the $\mathbf{J}$ and $\mathbf{K}$ would then be

$$\mathbf{J} = \bar{\mathbf{A}}\boldsymbol{\beta}\mathbf{Q}_\mathbf{H}\mathbf{\Delta}_\mathbf{H}^{-\frac{1}{2}} = \bar{\mathbf{A}}\mathbf{\Omega}, \text{ with } \mathbf{\Omega} := \boldsymbol{\beta}\mathbf{Q}_\mathbf{H}\mathbf{\Delta}_\mathbf{H}^{-\frac{1}{2}} \tag{24}$$

$$\mathbf{K} = \mathbf{\Delta}_\mathbf{H}^{\frac{1}{2}}\mathbf{Q}_\mathbf{H}^T. \tag{25}$$

Note that $\mathbf{H}$ can be rank deficient due to the lack of novel information in $\mathbf{C}$. In fact, $\mathbf{H}$ is always rank deficient by at least one due to centering of $\mathbf{C}$. Thus, the eigenvectors with zero eigenvalue should be ignored since they do not contribute towards describing $span(\mathbf{H})$. To do this, retain only the first $rank(\mathbf{M}_\mathbf{H})$ columns of $\mathbf{\Omega}$ and $rank(\mathbf{M}_\mathbf{H})$ rows of $\mathbf{K}$.

Matrix $\mathbf{F}$ is constructed as in (12) and diagonalized as $\mathbf{U}'\mathbf{\Sigma}'(\mathbf{V}')^T$. The left singular vectors of $[\hat{\mathbf{A}}^r\tilde{\mathbf{E}}]$ are

$$\mathbf{U}'' = [\mathbf{A}\boldsymbol{\alpha} \quad \bar{\mathbf{A}}\mathbf{\Omega}]\mathbf{U}' = \bar{\mathbf{A}}\mathbf{\Psi} \tag{26}$$

with $\mathbf{\Psi} := \begin{bmatrix}\boldsymbol{\alpha} \\ \mathbf{0}_{i,r}\end{bmatrix}\mathbf{U}'_{1:r,:} + \mathbf{\Omega}\mathbf{U}'_{(r+1):(r+rank(\mathbf{M}_\mathbf{H})),:}$. The first r kernel principal components of $[\mathbf{a}\,\mathbf{c}]$ are then revised as $\mathbf{U}^r \longleftarrow \bar{\mathbf{A}}\bar{\boldsymbol{\alpha}}$, where $\bar{\boldsymbol{\alpha}} := \mathbf{\Psi}_{:,1:r}$. Along with the overall data mean $\boldsymbol{\mu}_\mathbf{D}$, this is the updated KPCA. In addition, it is crucial to keep the corresponding singular values $\mathbf{\Sigma}^r = \mathbf{\Sigma}'_{1:r,1:r}$ for subsequent updates.

Note that there is no distinction between a *training* stage and an *update* stage. In fact, the training stage *is* the update stage. The procedure is actually started without having to know in advance how many vectors are to be processed. This feature is inherited from the underlying incremental PCA algorithm.

B. Maintaining Constant Update Speed

Up to this stage, the proposed algorithm is still exact. However, as shown by (9) and (26), it is unavoidable that the updated mean and kernel principal components have to be expanded from old and new data $[\mathbf{A}\,\mathbf{C}] = \phi([\mathbf{a}\,\mathbf{c}])$, where $[\mathbf{a}\,\mathbf{c}] \in \mathbb{R}^{m\times(n+i)}$. Although kernelizing incremental PCA spared us from computing a batch KPCA at every update, we still have to store all seen data. This is detrimental towards maintaining constant memory usage and update speed. To solve this problem, we compress the mean and kernel principal component representations by constructing RS expansions. We also propose an effective compression strategy to maximize approximation accuracy.

1) Constructing RS Expansions: Suppose we have a feature space vector $\mathbf{u} \in \mathcal{F}$ (e.g., kernel principal components) expanded from $(n+i)$ input data $\{\mathbf{x}_1, \cdots, \mathbf{x}_{(n+i)}\} \in \mathbb{R}^m$, i.e., $\mathbf{u} = \sum_{i=1}^{(n+i)}\bar{\alpha}_i\phi(\mathbf{x}_i)$. Here, $\phi : \mathbb{R}^m \longrightarrow \mathcal{F}$ is a mapping induced by a kernel function and $\bar{\boldsymbol{\alpha}} = [\bar{\alpha}_i \cdots \bar{\alpha}_{(n+i)}]^T$ are the expansion coefficients. We seek a *pre-image* $\mathbf{y} \in \mathbb{R}^m$ so that ideally $\mathbf{u} = \alpha\phi(\mathbf{y})$, with α being a constant. Most likely an exact pre-image will not exist [9], and we have to span $\mathbf{u}$ with a set of *approximate pre-images*, i.e., $\mathbf{u} \approx \sum_{i=1}^{p}\alpha_i\phi(\mathbf{y}_i)$, with $\boldsymbol{\alpha} = [\alpha_i \cdots \alpha_p]^T$ being the expansion coefficients for pre-images $\mathbf{y}_i$. Clearly, we are interested in the case where $p < (n+i)$ so that compression is achieved.

For the purpose of seeking a single approximate pre-image, it is sufficient to minimize the distance between $\mathbf{u}$ and its projection onto $\phi(\mathbf{y})$, which is equivalent to maximizing [20]

$$\frac{(\mathbf{u}\cdot\phi(\mathbf{y}))^2}{\phi(\mathbf{y})\cdot\phi(\mathbf{y})} = \frac{\left(\sum_{i=1}^{(n+i)}\bar{\alpha}_i k(\mathbf{x}_i,\mathbf{y})\right)^2}{k(\mathbf{y},\mathbf{y})}. \tag{27}$$

This can be performed using standard techniques [21], and for particular choices of kernels, fixed-point iterations can be used [9]. Once the optimum $\mathbf{y}$ is obtained, the constant α is set to $\alpha = (\mathbf{u}\cdot\phi(\mathbf{y}))/(\phi(\mathbf{y})\cdot\phi(\mathbf{y}))$. Refer to [6], [9] for details.

To construct RS expansions, i.e., to find $\{\mathbf{y}_1, \cdots, \mathbf{y}_p\}$, the process above is iterated. Given that the first approximate pre-image $\mathbf{y}_1$ is obtained, we repeat the process above to find a second pre-image $\mathbf{y}_2$ to reduce the approximation discrepancy,

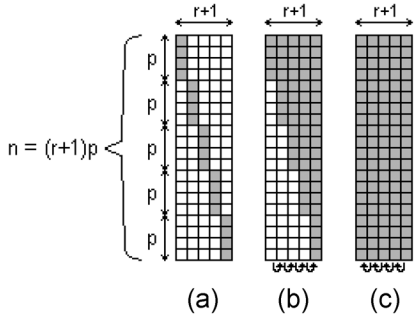


Fig. 1. Each block represents the matrix of linear expansion coefficients of a set of pre-images estimated for RS expansions of $r+1$ feature space vectors. Each square within a block indicates one coefficient, with grey squares representing nonzero coefficients and white otherwise. (a) Pre-images are inefficiently used. (b) Pre-images of a vector are used to aid in spanning a subsequent vector. (c) Second pass allows full use of pre-images.

i.e., $\alpha_2\phi(\mathbf{y}_2) = \mathbf{u} - \alpha_1\phi(\mathbf{y}_1)$. The error on which minimization is sought can be expanded linearly from feature space vectors

$$\phi(\mathbf{y}_k) = \sum_{i=1}^{(n+i)} \bar{\alpha}_i\phi(\mathbf{x}_i) - \sum_{j=1}^{k-1} \alpha_j\phi(\mathbf{y}_j) \tag{28}$$

and can be used directly in (27). The process is iterated until $k = p$. Naturally, the larger p is, the closer the RS expansion approximates the original feature space vector. At the kth iteration, the optimal expansion coefficients $\boldsymbol{\alpha} = [\alpha_1 \cdots \alpha_k]^T$ of the pre-images $\{\mathbf{y}_1, \cdots, \mathbf{y}_k\}$ to span the original feature space vector $\mathbf{u} = \sum_{i=1}^{(n+i)} \bar{\alpha}_i\phi(\mathbf{x}_i)$ can be computed as

$$\boldsymbol{\alpha} = (\mathbf{K}^{\mathbf{y}})^{-1}\mathbf{K}^{\mathbf{yx}}\bar{\boldsymbol{\alpha}} \tag{29}$$

with $\mathbf{K}^{\mathbf{y}}_{ij} := \phi(\mathbf{y}_i) \cdot \phi(\mathbf{y}_j)$ and $\mathbf{K}^{\mathbf{yx}}_{ij} := \phi(\mathbf{y}_i) \cdot \phi(\mathbf{x}_j)$. If $\mathbf{K}^{\mathbf{y}}$ does not have full rank, the pseudoinverse can be used. Refer to [6] and [9] for proofs and details.

2) Compressing Several Vectors Simultaneously: During each KPCA update, we have $(r+1)$ feature space vectors (r kernel principal components and one data mean) expanded from a library of $(n+i)$ mapped input data. By building RS expansions, we compress the representation of the feature space vectors by spanning them using p pre-images *each* (note that each feature space vector requires an RS expansion). Naturally, compression is worthwhile only if $(n+i) > (r+1)p$. To maintain nonincreasing memory usage and update duration, we keep p constant and use not more than $i = (n+i) - (r+1)p$ new input data to update the KPCA at every iteration, i.e., $n = (r+1)p$. The current library of $(n+i)$ expansion vectors is overwritten with $(r+1)p$ pre-images which will be used for subsequent updates. For each feature space vector, coefficients for unrelated pre-images can be set to zero. See Fig. 1(a).

The previous approach tends to be wasteful, since the mapped pre-images in the approximation of one feature space vector will unlikely be orthogonal to the other feature space vectors, and can be used essentially for free to aid in approximating the other vectors [6]. Therefore, if $\mathbf{u}_2$ is compressed after $\mathbf{u}_1$, these feature space vectors will have the form

$$\mathbf{u}_1 \approx \sum_{i=1}^{p} \alpha_i\phi(\mathbf{y}_i), \quad \mathbf{u}_2 \approx \sum_{i=1}^{p} \eta_i\phi(\mathbf{y}_i) + \sum_{j=p+1}^{2p} \eta_j\phi(\mathbf{y}_j) \tag{30}$$

where coefficients $\{\eta_1, \cdots, \eta_p\}$ are determined using (29) for $\{\mathbf{y}_1, \cdots, \mathbf{y}_p\}$ with respect to approximating $\mathbf{u}_2$, while new pre-images $\{\mathbf{y}_{(p+1)}, \cdots, \mathbf{y}_{2p}\}$ with coefficients $\{\eta_{(p+1)}, \cdots, \eta_{2p}\}$ are estimated, respectively, with iterating (27) and using (29), to further reduce the approximation discrepancy given by $\mathbf{u}_2 - \sum_{i=1}^{p} \eta_i\phi(\mathbf{y}_i)$. The pre-image set $\{\mathbf{y}_1, \cdots, \mathbf{y}_{2p}\}$ is then used to assist in compressing $\mathbf{u}_3$. Fig. 1(b) illustrates this process. After all $(r+1)$ feature space vectors are compressed, the final pre-image set $\{\mathbf{y}_1, \cdots, \mathbf{y}_{(r+1)p}\}$ supplants the original $(n+i)$-vector dataset.

We extend this concept further to incorporate a second pass of coefficient estimation. After completing the first stage of sequential forward compression, an extra boost of approximation accuracy is achievable if we re-estimate the coefficients for each feature space vector expansion with regards to the overall pre-image set. For example, for the kth feature space vector

$$\left\| \mathbf{u}_k - \sum_{i=1}^{(r+1)p} \sigma_i\phi(\mathbf{y}_i) \right\|_2 < \left\| \mathbf{u}_k - \sum_{i=1}^{kp} \eta_i\phi(\mathbf{y}_i) \right\|_2 \tag{31}$$

for $k < (r+1)$. Coefficients $\{\eta_1, \cdots, \eta_{kp}\}$ corresponding to pre-images $\{\mathbf{y}_1, \cdots, \mathbf{y}_{kp}\}$ which were estimated during forward compression is replaced with $\{\sigma_1, \cdots, \sigma_{(r+1)p}\}$ obtained using (29) from the overall pre-image set. Although without theoretical guarantee, in practice an improvement over the first pass of compression is almost always achieved. This is despite most of the pre-images not being specifically tailored to span a particular feature space vector. See Fig. 1(c).

A positive side effect to using our compression strategy is that since RS expansions are built repetitively, when the whole dataset has been processed the resulting kernel principal component expansions are automatically sparse. This is desirable for applications that require fast utilization of KPCA results. Also, note that one could modify (27) and attempt to minimize the joint-distance between a set of feature space vectors and the projections these feature space vectors onto a pre-image, i.e., to estimate a *shared* pre-image across all feature space vectors [20]. This is then iterated to build the desired pre-image set. Through experiments, given a number of total pre-images $(r+1)p$ to obtain for a KPCA basis and mean, we found that this method on average produces inferior approximations than if the pre-images are estimated for each feature space vector individually. This is probably due to the fact that a shared pre-image has to approximate all feature space vectors simultaneously, hence the pre-image does not satisfactorily approximate any one of feature space vectors.

C. Enforcing the Orthogonality of KPCA Basis

Most likely a compressed basis $\tilde{\mathbf{U}}^r = \mathbf{Y}\boldsymbol{\sigma}$ obtained by constructing RS expansions to approximate the original KPCA basis $\mathbf{U}^r = \bar{\mathbf{A}}\bar{\boldsymbol{\alpha}}$ from (26) is nonorthogonal due to approximation errors. Here, $\mathbf{Y}$ is the matrix of the mapped pre-image set $[\phi(\mathbf{y}_1) \cdots \phi(\mathbf{y}_{(r+1)p})]$ which has coefficients $\boldsymbol{\sigma} \in \mathbb{R}^{(r+1)p \times r}$ that are estimated to closely span the original kernel principal components. To re-orthogonalize $\tilde{\mathbf{U}}^r$ we can use kernel SVD

again. We compute the kernel matrix associated with $\tilde{\mathbf{U}}^r$ given by

$$\mathbf{M}_o = \boldsymbol{\sigma}^T \mathbf{Y}^T \mathbf{Y} \boldsymbol{\sigma} \tag{32}$$

with eigenvector decomposition $\mathbf{M}_o = \mathbf{Q}_o \mathbf{D}_o \mathbf{Q}_o^T$. The orthogonalized basis is then $\mathbf{Y}\boldsymbol{\sigma}\mathbf{Q}_o\mathbf{D}_o^{-(1/2)}$ on which we project the approximated basis via

$$\mathbf{P} = \left(\boldsymbol{\sigma}\mathbf{Q}_o\mathbf{D}_o^{-\frac{1}{2}}\right)^T \mathbf{Y}^T\mathbf{Y}\boldsymbol{\sigma}. \tag{33}$$

The orthogonal basis $\mathbf{Y}\boldsymbol{\sigma}\mathbf{Q}_o\mathbf{D}_o^{-(1/2)}$ cannot be used directly because the ordering of the kernel principal components could have been compromised due to nonuniform vector lengths caused by RS approximations. Thus, the compressed basis is expanded using the orthogonal basis as

$$\tilde{\mathbf{U}}_o^r = \mathbf{Y}\boldsymbol{\sigma}\mathbf{Q}_o\mathbf{D}_o^{-\frac{1}{2}}\mathbf{P} = \mathbf{Y}\boldsymbol{\pi} \tag{34}$$

where $\boldsymbol{\pi} := \boldsymbol{\sigma}\mathbf{Q}_o\mathbf{D}_o^{-(1/2)}\mathbf{P}$. To normalize the basis vectors, the columns of $\mathbf{P}$ are normalized. To express the singular values corresponding to the new basis, project the original singular values onto this basis via

$$\mathbf{S} = \left(\tilde{\mathbf{U}}_o^r\right)^T \mathbf{U}^r\boldsymbol{\Sigma}^r = \boldsymbol{\pi}^T\mathbf{Y}^T\bar{\mathbf{A}}\bar{\boldsymbol{\alpha}}\boldsymbol{\Sigma}^r. \tag{35}$$

The singular values associated with the new basis is $\tilde{\boldsymbol{\Sigma}}^r = diag(\mathbf{S})$. The nonzero off-diagonal elements of $\mathbf{S}$ discarded due to the $diag(\cdot)$ operation correspond to the approximation errors of the RS approximation. Note that the process does not require explicit evaluations of the mapping ϕ since only dot products between feature space vectors are required.

TABLE I
PSEUDOCODE FOR INCREMENTAL KPCA

Incremental KPCA with RS compressions

Input: Training data. Value for parameters r, p and i.

Output: Kernel principal components (KPCs) and data mean.

```
1. Obtain (r + 1)p data and perform batch KPCA.
2. Retain r KPCs and data mean.
3. Initialize library with data.
4. while Data is not exhausted do
5.     Obtain i new data.
6.     Update KPCs and data mean (§IV).
7.     Append new data to library.
8.     if Number of stored data > (r + 1)p then
9.         Compress KPCs and mean with p pre-images each.
10.        Enforce orthogonality of KPC basis.
11.        Overwrite data library with pre-image set.
12.        Update KPC expansion coefficients.
13.    end if
14. end while
```

D. Selecting Parameters

Table I lists the overall pseudocode for the proposed incremental KPCA procedure. The list of parameters required for the method are as follows.

1) r: The number of kernel principal components to maintain.
2) p: The number of vectors/preimages to estimate/store per feature space vector.
3) i: The number of increment vectors to use per update.

The first step involves obtaining $(r+1)p$ vectors to perform an initial KPCA from which r kernel principal components are obtained. The steps of incremental KPCA which involve updating and compressing then begin and are repeated until all input vectors are exhausted.

Parameters r and i arose from the underlying incremental PCA algorithm, hence previous works on incremental PCA such as [22]–[24] can provide useful insights on how these parameters can be determined. Ideally, parameter r is allowed to depend on the intrinsic characteristics of the input data, i.e., if the data distribution can be parsimoniously described by the first few kernel principal components, then r can be set accordingly to that number. In most realistic cases, the eigenspectrum of the data decays exponentially, i.e., the data is not confined to a *flat* hyperplane. Hence what can be done is to set the value of r to reflect the balance between maintaining the compactness and accuracy of the KPCA model. Also, through experiments we find that setting r to more than what is desired helps in minimizing inaccuracies since at the initial stages it is unknown which vectors are aligned with the true dominant directions of variation and which are merely noise. Parameter i affects directly the total number of updates incremental KPCA will perform. Reducing the number of updates by setting i to a large value helps in minimizing the inaccuracy of the eventual KPCA model [23], but this translates into more time per update.

Parameter p influences the approximation accuracy of RS compressions as well as the speed of IKPCA-RS. Unfortunately these are contrasting needs, since increased accuracy entails more pre-images and hence decreased speed, and vice versa. A balance must be struck between these two requirements. At this stage we do not have a general rule of determining p for any particular dataset, but through experiments we found $p = 10$ to be satisfactory for our purposes. Certainly as RS estimation methods improve in terms of speed, p can be set to a bigger value to increase the ratio of approximation accuracy per update duration.

E. Analysis of Complexity

By examining the proposed algorithm in Section IV, we can see that each update is dependent only on the incremental data $\mathbf{c}$ and the data library $\mathbf{a}$ whose size is kept constant through RS compressions. The size of $\mathbf{c}$ and the number of kernel principal components are usually maintained the same throughout processing the overall dataset. However, the construction of RS

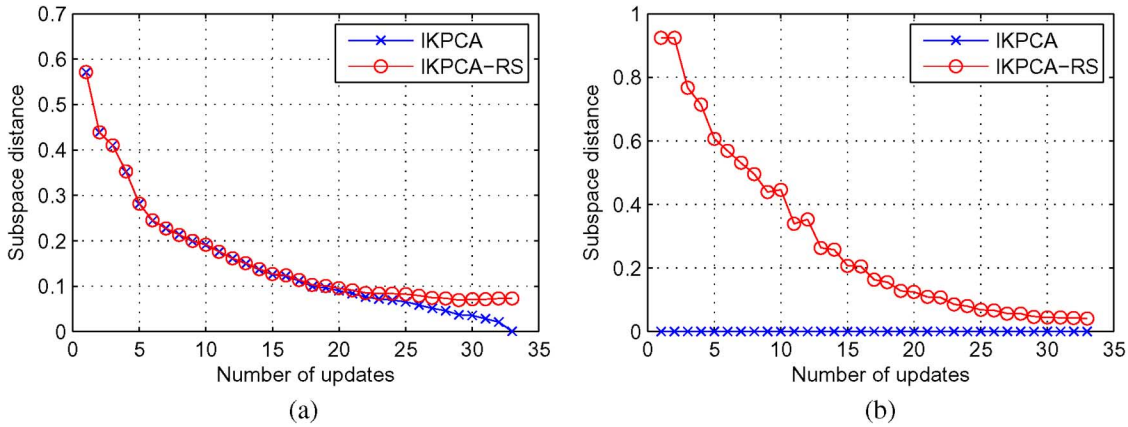


Fig. 2. Examining the severity of drift of IKPCA-RS. The curves indicate results averaged over 100 repetitions. (a) Gaussian kernel with $\sigma = 1$. (b) Polynomial kernel of degree 2.

compressions is iterative in nature, so there is no guarantee that each feature space vector can be compressed in the same amount of time. However, as verified by the experiments in Section V, for a given p and i, each compression does take approximately the same time, so the proposed algorithm has linear time complexity.

It is also interesting to see how the duration for a *single* update scales according to the parameters r, p and i. Appendix I provides a summary of the time complexity involved for an update. In particular, the eigendecomposition of matrices $\mathbf{M_H}$ and $\mathbf{F}$ can potentially be time-consuming with complexity $\mathcal{O}(i^3)$ and $\mathcal{O}(r^3)$. However, the values of i and r are often tiny ($0 < i$, $r < 100$) relative to the size of the overall dataset (more than 3000 to make incremental computations attractive). As mentioned previously, p is usually set to 10. The commonly used kernel functions require constant evaluation time, and this scales with the dimension of the input space m with the exact time depending on the specific kernel type. Second, it can be seen from Appendix I that since all r, p and i are tiny compared to the overall dataset size, the proposed incremental KPCA algorithm requires only a small fraction of the memory needed for batch KPCA computations.

V. Experimental Results

Several experiments were conducted to examine three properties of the proposed incremental KPCA algorithm: A) the accuracy of the proposed method in approximating batch KPCA computation; B) empirical time complexity of the proposed method; C) effectiveness of the proposed method in practical applications. For the following, we define IKPCA-RS as the acronym of the proposed method, while IKPCA simply means the proposed method without RS compressions. KPCA means the traditional batch computation method. All programs were executed in Matlab on a Microsoft Windows-based PC.

A. Approximation Accuracy

One major drawback of incremental computations is the danger of drift, i.e., error accumulates across the update iterations causing the discrepancy between the incremental results and the ground truth results (computed using batch methods) to grow indefinitely large. In the case of IKPCA-RS, discrepancies produced from RS compressions is the major source of error. The first experiment serves to examine the severity of drift of incremental IKPCA-RS.

1) Using Synthetic Datasets: One hundred 2-D synthetic datasets of 1000 vectors each were randomly generated in the following manner: x-values have uniform distribution in [−1,1], while y-values are generated from $y = x^2 + \xi$, where ξ is normal noise with standard deviation 0.2. The datasets were processed as the following, with the training points uniformly sampled:

- KPCA with $r = 3$;
- IKPCA with $r = 6$, $i = 30$;
- IKPCA-RS with $r = 6$, $i = 30$, $p = 10$.

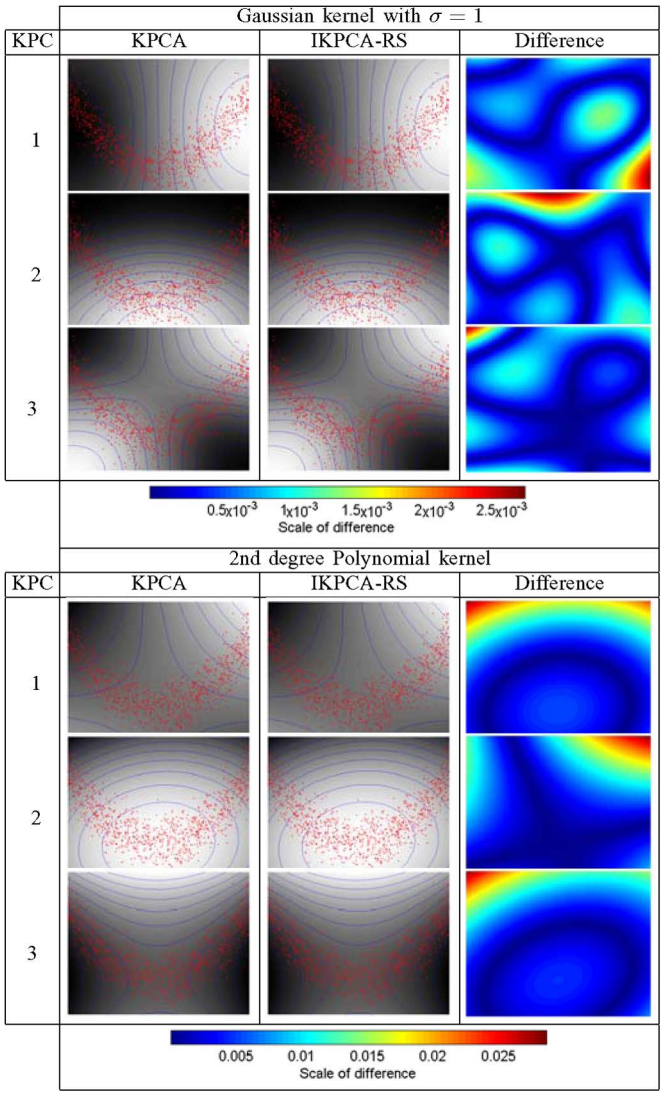
At each update iteration, the distance between the subspaces (in RKHS) spanned by the the kernel principal components of the incremental methods and KPCA was computed. We constructed a simple distance measure based on the (kernel) principle angles [25] between subspaces

$$d(\mathcal{X}, \mathcal{Y}) = \sqrt{\sum_{i=1}^{r} \theta_i^2} \tag{36}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are r-dimensional subspaces (in RKHS) and $\{\theta_1, \cdots, \theta_r\}$ are the (kernel) principal angles between them. The distance is bounded by $\sqrt{r}\pi/2$ and it reduces to 0 when two subspaces are identical. Only 3-D subspaces (from the first three kernel principal components) were compared—r was set to 6 for the incremental methods to minimize error of projecting information away during updating. The Gaussian kernel with $\sigma = 1$ and second-degree polynomial kernel were used. Fig. 2 illustrates the results.

It can be seen that initially subspaces from IKPCA-RS gradually converged toward the ground truth subspaces, but eventually they remained at a stable distance apart and did not appreciably diverge from the ground truth subspaces. Hence, it can be concluded that no "growing" drift occurred. Also, the mean of the data from IKPCA-RS and KPCA were also compared by computing the angular difference, and similar curves as those in Fig. 2 were obtained. In the Gaussian kernel case, the subspaces of IKPCA actually almost converged to their respective ground truth subspace. For the polynomial kernel case, since the RKHS

TABLE II
COMPARISON OF PROJECTION VALUES OF KPC



is only 6-D, setting $r = 6$ for IKPCA means the data can be perfectly reconstructed by the kernel principal components, hence no approximation error occurred.

Perhaps of more practical importance is the effect of the difference between the kernel principal components in the input space. For one of the generated datasets, the subspace distance between the subspaces of KPCA and IKPCA-RS is about 0.07 (Gaussian) and 0.08 (polynomial). The projection values of points sampled uniformly in the 2-D input space, computed using (7), are compared. Table II illustrates the results: Red dots are the input vectors, while blue contour lines and background shading depict the projection values. It can be seen that differences in the projection values are very small, hence it can be concluded that results from IKPCA-RS follow the ground truth results closely.

2) Processing Image Datasets: The previous experiment was repeated for datasets of real images. We made use of the following four datasets: a) the Teapots dataset which contains 400 images of a teapot undergoing a full rotation; b) 1000 images of the digit "3" from the MNIST handwritten digits dataset; c) the Freyface database which contains 1965 images of a video recording of a face with changing pose and expression; d) the Yale Face Database B [26] which contains 5760 images of faces in different poses and lighting conditions. IKPCA-RS parameters are $r = 20$, $p = 10$ and $i = 30$, with the increment vectors drawn randomly. Despite maintaining the first 20 kernel principal components, only the first 6 are compared against the ground truth kernel principal components from KPCA using (36).

Fig. 3 illustrates the results. It can be seen that the eventual discrepancy of IKPCA-RS from KPCA varies between the datasets. Some are as high as 2.0 subspace distance, while some are as low as 0.2 subspace distance. Also, the difference between IKPCA and IKPCA-RS illustrates how much error the RS compressions have caused, and this varies as well between the datasets. Certainly, the kernel function and parameter plays a part too since these affect the distribution of the datasets in the RKHS. Nonetheless, the curves exhibit that no drift has occurred for IKPCA-RS. An objective comparison between the results of KPCA, IKPCA, and IKPCA-RS is by using the average residual error of training data points in the RKHS (distances between the data points and their projections onto the kernel principal components); refer to [7] on how this can be computed. These are summarized in Table III, where it can be seen that the average residual error of all three methods are similar. Nonetheless, whether the discrepancy of IKPCA-RS from the ground truth is acceptable depends on the intended application of the kernel principal components.

B. Empirical Time Complexity

Here, we examine the empirical time complexity of IKPCA-RS. First, the 2-D synthetic dataset used in Section V-A was created in various sizes and processed with IKPCA-RS with parameters $r = 3$, $p = 10$, and $i = 30$. The Gaussian kernel with $\sigma = 1$ and the second-degree polynomial kernel were used. Secondly, images of digit "3" from the MNIST database were used by creating subsets of various sizes by random sampling. The subsets were then processed with IKPCA-RS with parameters $r = 5$ and 20, $p = 10$, and $i = 30$. Again, the Gaussian kernel with $\sigma = 250$ and the second-degree polynomial kernel were used. The elapsed time for each IKPCA-RS instance was recorded. The processing times of batch KPCA for various sizes of the 2-D synthetic dataset were also obtained. Processing times of KPCA are independent of the number of kernel principal components maintained, type of data, and kernel; hence, these can be used directly for comparisons with IKPCA-RS times for the MNIST images.

Fig. 4 shows the results which confirm that processing time for IKPCA-RS scales linearly with the number of data. Also, as expected KPCA scales with complexity $\mathcal{O}(n^3)$. It can be seen that processing times for IKPCA-RS vary depending on the type of data (i.e., the length of the input vectors), type of kernel function and the number of kernel principal components maintained (see Section IV-E and Appendix I). Also, at this stage performing batch KPCA is more attractive than IKPCA-RS for small scale datasets—IKPCA-RS requires approximately 800 s to process 1000 images from the MNIST database, while KPCA

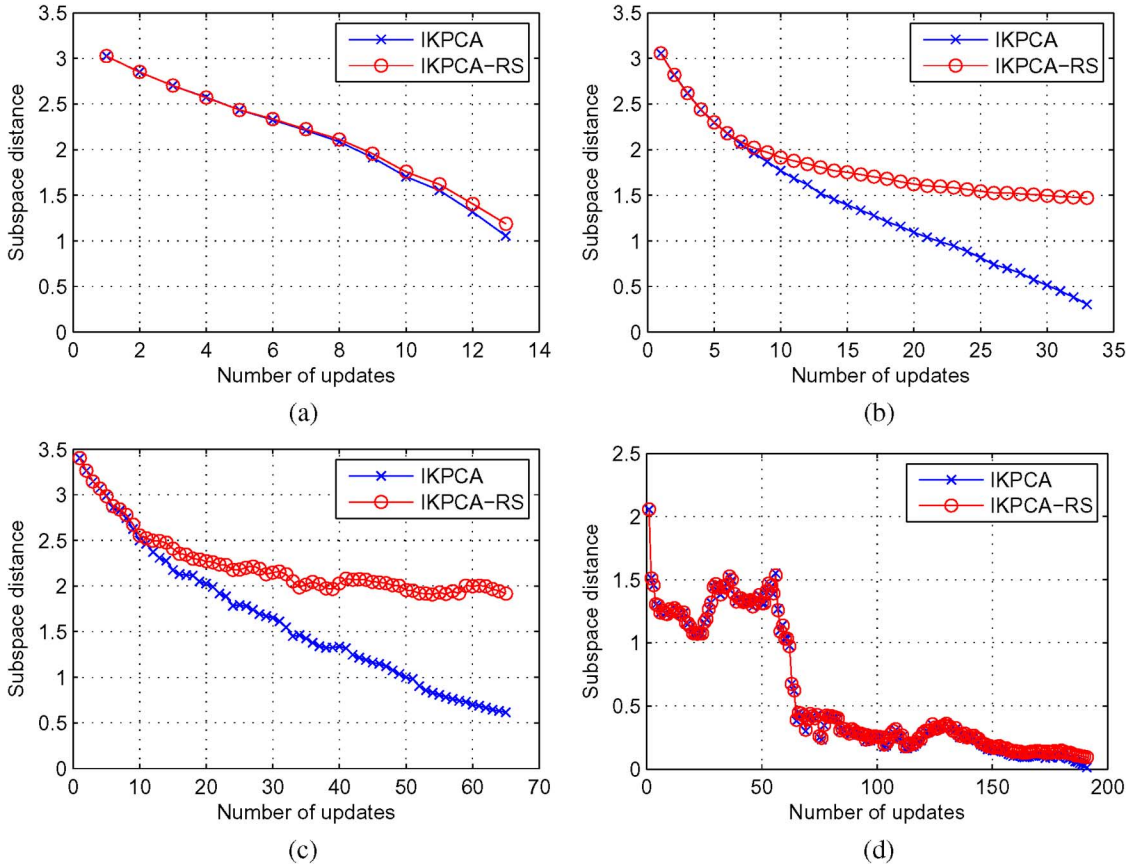


Fig. 3. Progression of the kernel principal components across the updates. The curves are averaged from N repetitions, with $N = 100$ for (a) and (b), $N = 20$ for (c), and $N = 1$ for (d). (a) Teapots dataset, Gaussian kernel with $\sigma = 100$; (b) digit "3" of MNIST database, Gaussian kernel with $\sigma = 250$; (c) Freyface dataset, Gaussian kernel with $\sigma = 100$; (d) Yale Face Database B, second-degree polynomial kernel.

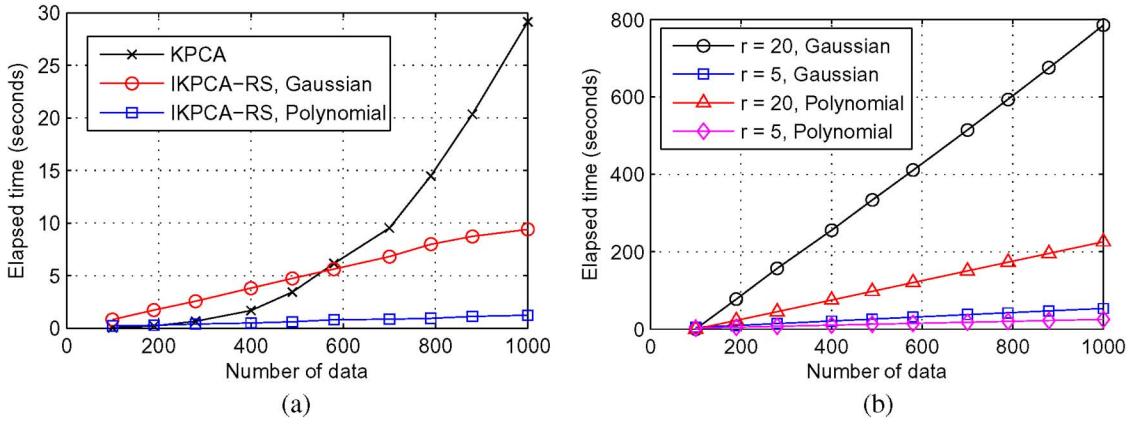


Fig. 4. Processing times of IKPCA-RS and KPCA. All curves were obtained by averaging results from 20 repetitions: (a) 2-D synthetic dataset; (b) digit "3" images (all processed with IKPCA-RS).

TABLE III
AVERAGE RESIDUAL ERROR OF THREE KPCA METHODS

	KPCA	IKPCA	IKPCA-RS
Teapots	1.4841	1.4744	1.4481
Digit "3"	1.1247	1.1241	1.0917
Freyface	1.0953	1.0939	1.0672
Yale Faces	2.6272×10^{13}	2.6272×10^{13}	2.6265×10^{13}

requires only about 30 s. Nonetheless, for much larger datasets (e.g., more than 10 000 vectors), it is expected that IKPCA-RS will be the more viable alternative. Most of the time in each update duration of IKPCA-RS is used for performing RS compressions, hence improving the implementation or method of pre-image estimation can directly speed up IKPCA-RS. Note that pre-image estimation is still a relatively new area within the kernel literature [6].

C. Practical Applications

We demonstrate three applications for IKPCA-RS. The first application requires processing massive amounts of images, and, thus, it benefits from a lowered memory requirement

TABLE IV
KERNEL EIGENFACES CLASSIFICATION RESULTS

Algorithm	Training images	Kernel	Error rate (%)
KPCA	1000 (Set 3)	rbf, $\sigma = 1$	65.92 (501/760)
IKPCA-RS	1000 (Set 3)	rbf, $\sigma = 1$	66.84 (508/760)
KPCA	1000 (Set 3)	poly, $d = 2$	49.61 (377/760)
IKPCA-RS	1000 (Set 3)	poly, $d = 2$	50.26 (382/760)
IKPCA-RS	5000 (Set 1)	rbf, $\sigma = 1$	46.58 (354/760)
IKPCA-RS	5000 (Set 1)	poly, $d = 2$	26.97 (205/760)

and time complexity afforded by substituting batch KPCA with IKPCA-RS. The second application requires on-the-fly processing of images obtained from a video stream, and, thus, it serves to exhibit the advantage of being able to process KPCA in an online manner (a capability unavailable in methods like [2] and [8] as explained in Section II). The third application serves to investigate the feasibility of computing IKPCA-RS in a nonstationary environment, i.e., the KPCA model is required to adapt to accurately describe recent observations only. Without modifications, this cannot be performed using [2] and [8].

1) Face Recognition: The kernel eigenfaces method [1] for face recognition was implemented on the Yale Face Database B [26] which contains 5760 images of ten subjects in multiple poses and lighting conditions. The images were cropped and resized to 20 $\times$ 20 pixels before being normalized to [0, 1] in intensity. The dataset was then randomly partitioned into two disjoint subsets: Set 1 contained 5000 training images, and Set 2 contained 760 test images. Another set called Set 3 was created by randomly choosing 1000 images from Set 1. Table IV summarizes the training schemes that were carried out. The kernels were chosen based on [1]. By training Set 1 with IKPCA-RS (with parameters $r = 36$, $p = 10$, and $i = 30$), we avoided storing a massive 5000 $\times$ 5000 kernel matrix for KPCA (the largest intermediate matrix for IKPCA-RS is $\Omega \in \mathbb{R}^{40\times 31}$). For classification, the training and testing images were projected onto the first-36 kernel principal components and the k-nearest neighbour rule with $k = 10$ was evaluated. The results in Table IV show that by being able to process more exemplars, the kernel principal components obtained using IKPCA-RS from Set 1 managed to outperform those estimated via KPCA on Set 3. Moreover, the performances of IKPCA-RS and KPCA on Set 3 were almost identical. Despite the modest accuracy compared to other face recognition methods, what is demonstrated here is the benefit of being able to process large datasets, as well as the approximation accuracy of IKPCA-RS. In [1], kernel eigenfaces yielded a similar error rate of 27.27% (45/165) on the much smaller and less challenging Yale Face Database.[1]

2) Online Face Recognition From Video for Access Control: This requires face images from video streams to be processed on-the-fly and decisions to be made quickly for the convenience of the user. Certainly, online processing also relieves the demand for the amount of memory required, especially if the algorithm is implemented on a low-cost non-PC based system. We implemented the kernel mutual subspace (KMS) method [11] which essentially works by building a subspace for a set of images of the same face (from a video sequence) in a kernel-induced feature space, and identities are distinguished by comparing subspaces using distance measures such as (36) in RKHS. Performing face recognition in such a manner was also proposed in [25]. The subspaces are spanned by the principal (kernel) eigenvectors of the face image set estimated using KPCA *without* a prior mean adjustment. Therefore, when IKPCA-RS is used to perform *online* KMS, the mean updating steps in Section IV can be discarded and matrix $\tilde{\mathbf{E}}$ in (11) is simply replaced with the increment images $\mathbf{C}$ (basically, this is what was proposed in our previous work [12], but we improve it with the enhancements introduced in this paper).

Fig. 5. Examples of preprocessed images from the face video database. Note the variation in pose, expression, and lighting that affects the images. Also, a lot of background clutter is included by the face detector.

A 17-subject face video database was collected. Each subject has ten image sets of about 100 images each. The AdaBoost face detector [27] was applied to segment the faces; Fig. 5 illustrates. One image set per subject was used for training, while the rest were used for testing (allowing a total of 153 tests). Subspaces of five-dimensions were built to represent the face image sets using IKPCA-RS, while batch KCPA and linear PCA [this is called the mutual subspace method (MSM) [28]] were also carried out for benchmarking purposes. For the kernel methods, the Gaussian kernel with $\sigma = 300$ was used. The parameters for IKPCA-RS are $r = 5$, $p = 10$, and $i = 5$. From the ROC curves in Fig. 6(a), it can be seen that IKPCA-RS results follow KPCA results closely. Also, the kernel methods performed better than linear PCA. Fig. 6(b) depicts how the subspaces estimated using IKPCA-RS "move" with reference to the ground-truth subspaces from KPCA. No visible drift can be observed, while the final discrepancy between IKPCA-RS and KPCA will not significantly affect the recognition performance since it is far below the average within-class distance. Our implementation (unoptimised) currently runs at about 5 frames/s.

3) Visual Tracking: In [10], incremental PCA was used to adapt appearance models of objects built using PCA to perform visual tracking. We extend this concept to allow visual tracking using KPCA appearance models which is adapted using IKPCA-RS. Appearance modeling by KPCA provides the opportunity of utilizing specially crafted kernels for specific objects in the task of tracking. For example, see [29] and [30]. Given a KPCA model which encodes the *recent* appearance of the object of interest up to the *previous* frame, candidate subimages of the *current* frame which could potentially enclose the object of interest are tested by computing their orthogonal distance to the subspace spanned by the kernel principal components. Note that mean-adjustment is required here. The subimage with the closest matching appearance is resized to a

[1]Available at http://www.vision.ucsd.edu/kriegman-grp/software.html. Note that we are not trying to present the best face recognition method. Our aim is to show how an existing KPCA-based method can be improved by allowing it to process larger datasets.

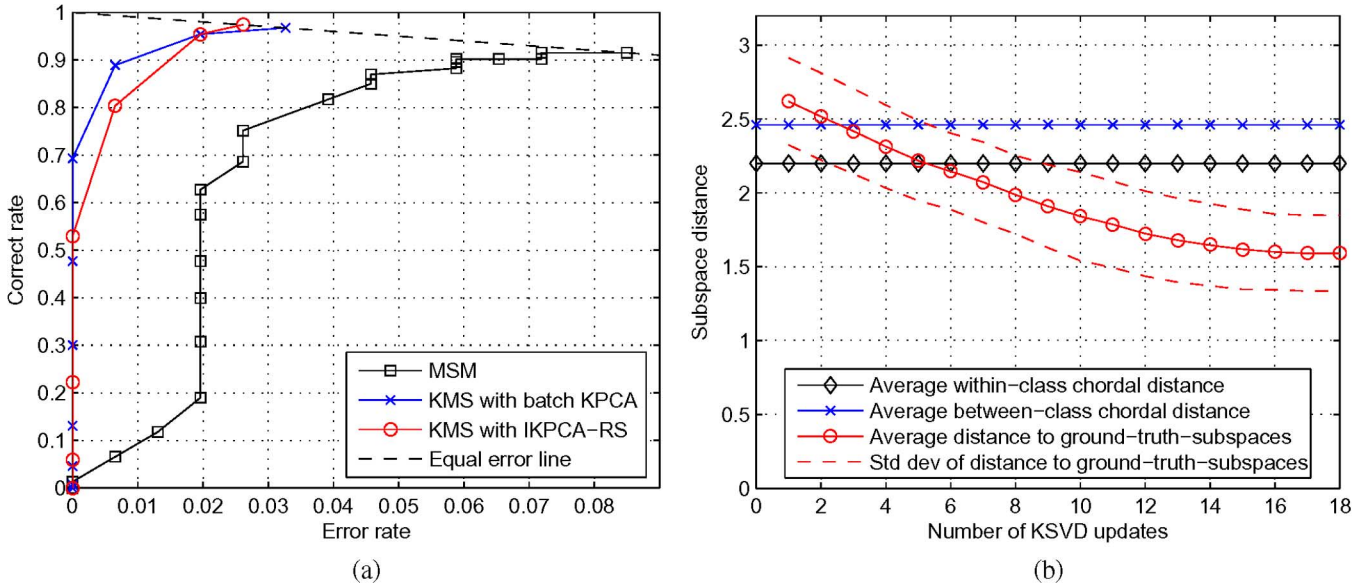


Fig. 6. Online face recognition from video results. For (b), the curves are averaged across subspaces estimated for the training and testing image sets. (a) ROC curves of KMS and MSM. (b) Face subspace update progression.

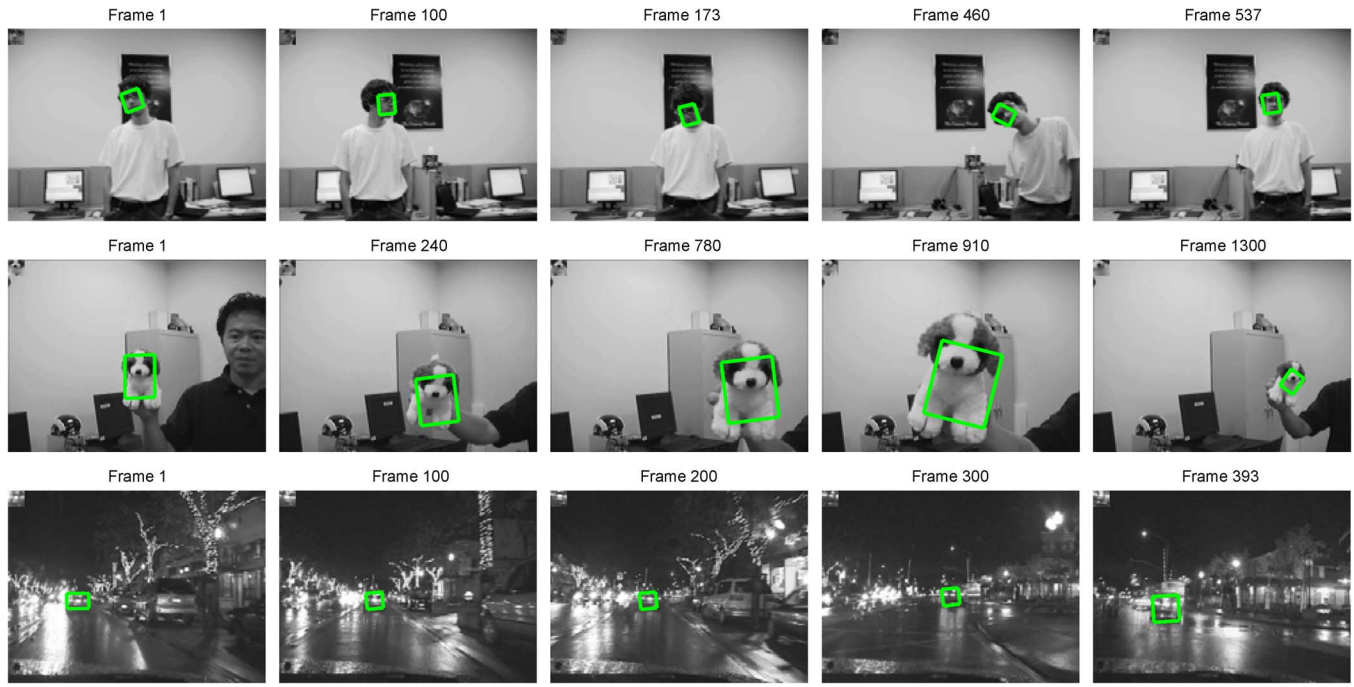


Fig. 7. First sequence involves tracking a face which undergoes large pose changes and translations. The second sequence involves tracking a soft-toy which undergoes swift movement with motion blur and large scale-variation. The third sequence involves tracking the back of a vehicle on a busy street at night. IKPCA-RS managed to track the objects of interest throughout the whole of their respective sequences.

fixed size and used to update the KPCA model. Fig. 7 shows IKPCA-RS tracking results of three video sequences.[2]

In our experiments, KPCA updates are performed after every five tracks; hence, the parameters for IKPCA-RS are $r = 5$, $p = 10$, and $i = 5$. The compression stage in IKPCA-RS allows tracking to be performed with nonincreasing durations across frames. On average, IKPCA-RS runs at about 5 frames/s. We followed the approach of [31] to generate the candidate subimages: A fixed-shaped Gaussian distribution is placed over the position and orientation of the previous tracking rectangle, and 200 candidate subimages are drawn based on the Gaussian—this basically means that tracking depends *mainly* on the effectiveness of the appearance model. Sophisticated motion estimation like particle filtering [10] can be employed to aid in tracking of sequences with more challenging trajectories.

[2]Available at http://www.cs.toronto.edu/~dross/.

VI. Conclusion and Future Work

In this paper, we have proposed a method for incrementally computing KPCA. This is useful for processing large datasets

TABLE V
MEMORY SIZE AND TIME COMPLEXITY FOR A SINGLE KPCA UPDATE

Matrix/Process	Memory size	Time complexity
$\mathbf{L}$	$r(i+1)$	$\mathcal{O}(n^2)$, $\mathcal{O}(i^2)$, $\mathcal{O}(r)$
$\boldsymbol{\beta}$	$(n+i)(i+1)$	$\mathcal{O}(n)$, $\mathcal{O}(i^2)$
$\mathbf{M_H}$	$(i+1)^2$	$\mathcal{O}(n^2)$, $\mathcal{O}(i^3)$
$eig(\mathbf{M_H})$	–	$\mathcal{O}(i^3)$
$\boldsymbol{\Omega}$	$(n+i)(i+1)$	$\mathcal{O}(n)$, $\mathcal{O}(i^3)$
$\mathbf{K}$	$(i+1)^2$	$\mathcal{O}(i^3)$
$\mathbf{F}$	$(r+i)(r+i+1)$	–
$eig(\mathbf{F})$	–	$\mathcal{O}(r^3)$, $\mathcal{O}(i^3)$
$\boldsymbol{\Psi}$	$(n+i)(r+i)$	$\mathcal{O}(n)$, $\mathcal{O}(i^3)$, $\mathcal{O}(r^2)$

and potentially for online computations. Through theoretical analyses and experimental results it is shown that the proposed method has linear time complexity, much better than the standard approach which involves a complexity of $\mathcal{O}(n^3)$. Furthermore, the method can easily update a previous KPCA computation with novel data. Experimental results show that the method can accurately approximate standard KPCA computations. Several practical applications using the proposed method were also demonstrated. Future work include improving our implementation of the proposed method, especially the RS compression stage, to enable real-time processing of data. Second, we wish to investigate using noniterative methods for constructing RS expansions such as [32] which can potentially improve the speed and accuracy of RS compressions.

APPENDIX I
MEMORY REQUIREMENT AND TIME COMPLEXITY

Table V lists the memory requirement and time complexity involved for performing a single KPCA update (see Section IV). As an example of how we derive the complexity results, consider obtaining matrix $\mathbf{L} = \boldsymbol{\alpha}^T\mathbf{A}^T\bar{\mathbf{A}}\boldsymbol{\gamma}$. Computing $\mathbf{A}^T\bar{\mathbf{A}} = \phi(\mathbf{a})^T\phi([\mathbf{a}\ \mathbf{c}])$ requires $(n+i)n$ kernel evaluations. Producing $\boldsymbol{\alpha}^T\mathbf{A}^T\bar{\mathbf{A}}$ involves merely matrix multiplications, and with $\boldsymbol{\alpha} \in \mathbb{R}^{n\times r}$ and $\mathbf{A}^T\bar{\mathbf{A}} \in \mathbb{R}^{n\times(n+i)}$, this requires $2rn(n+i)$ computations. With $\boldsymbol{\alpha}^T\mathbf{A}^T\bar{\mathbf{A}} \in \mathbb{R}^{r\times(n+i)}$ and $\boldsymbol{\gamma} \in \mathbb{R}^{(n+i)\times(i+1)}$, the final step $\boldsymbol{\alpha}^T\mathbf{A}^T\bar{\mathbf{A}}\boldsymbol{\gamma}$ involves $2r(n+i)(i+1)$ computations. The total computational effort required is

$$(n+i)\left(nK_o + 2r(n+i+1)\right) \tag{37}$$

with K_o being the computational effort required for one kernel evaluation. K_o is a function of input space dimensionality, m, and the type of kernel used. From (37), for a fixed kernel, the time for computing $\mathbf{L}$ scales with regards to the independent variables n, i, and r as $\mathcal{O}(n^2)$, $\mathcal{O}(i^2)$ and $\mathcal{O}(r)$.

ACKNOWLEDGMENT

The authors would like to thank V. Franc of the Center for Machine Perception, Czech Technical University, Prague, for making publicly available the Statistical Pattern Recognition Toolbox, of which their experiments made use. They would also like to thank the reviewers for their helpful comments.

REFERENCES

[1] M.-H. Yang, "Kernel eigenfaces vs. kernel fisherfaces: Face recognition using kernel methods," in *Proc. 5th IEEE Conf. Automatic Face and Gesture Recognition*, 2002, pp. 215–220.
[2] K. I. Kim, M. O. Franz, and B. Schölkopf, "Iterative kernel principal component analysis for image modeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 9, pp. 1351–1366, Sep. 2005.
[3] S. Mika, B. Schölkopf, A. Smola, K.-R. Müller, M. Scholz, and G. Rätsch, "Kernel PCA and de-noising in feature spaces," presented at the Advances in NIPS, 1998.
[4] J. Meltzer, M.-H. Yang, R. Gupta, and S. Soatto, "Multiple view feature descriptors from image sequences via kernel PCA," in *Proc. ECCV*, 2004, pp. 215–227.
[5] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Comput.*, vol. 10, pp. 1299–1319, 1998.
[6] B. Schölkopf and A. Smola, *Learning Eith Kernels*. Cambridge, MA: MIT Ppress, 2002.
[7] N. Cristianini and J. Shawe-Taylor, *Kernel Methods for Pattern Analysis*. New York: Cambridge Univ. Press, 2004.
[8] B.-J. Kim, "Active visual learning and recognition using incremental kernel PCA," in *Proc. Austral. Conf. on Artificial Intelligence*, 2005, pp. 585–592.
[9] B. Schölkopf, S. Mika, C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola, "Input space vs feature space in kernel-based methods," *IEEE Trans. Neural Netw.*, vol. 10, no. 5, pp. 1000–1017, May 1999.
[10] J. Lim, D. Ross, R.-S. Lin, and M.-H. Yang, "Incremental learning for visual tracking," presented at the Advances in NIPS, 2004.
[11] H. Sakano, N. Mukawa, and T. Nakamura, "Kernel mutual subspace method and its application for object recognition," *Electron. Commun. Jpn.*, vol. 2/88, 2005.
[12] T.-J. Chin, K. Schindler, and D. Suter, "Incremental kernel SVD for face recognition with image sets," presented at the 7th IEEE Conf. Automatic Face and Gesture Recognition, 2006.
[13] V. Franc and V. Hlaváč, "Greedy algorithm for a training set reduction in the kernel methods," in *Proc. Int. Conf. Computer Analysis of Images and Patterns*, 2003, pp. 426–433.
[14] V. Franc, "Optimization algorithms for kernel methods," Ph.D. dissertation, Center for Machine Perception, Czech Tech. Univ., Prague, 2005.
[15] C. K. I. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," presented at the Advances in NIPS, 2001.
[16] A. J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," presented at the Int. Conf. Machine Learning, 2000.
[17] S. Fine and K. Scheinberg, "Efficient SVM training using low-rank kernel representation," Tech. Rep., Res. Rep. RC 21911, IBM T. J. Watson Research Center, Yorktown Heights, NY, 2000.
[18] A. Frieze, R. Kannan, and S. Vempala, "Fast Monte-Carlo algorithms for finding low-rank approximations," in *Proc. Conf. Foundations of Computer Science*, 1998, pp. 370–378.
[19] M. Brand, "Incremental singular value decomposition of uncertain data with missing value," presented at the ECCV, 2002.
[20] B. Schölkopf, P. Knirsch, A. Smola, and C. Burges, "Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces," in *Proc. DAGM Symp.*, 1998, pp. 124–132.
[21] C. Burges, "Simplified support vector decision rules," in *Proc. Int. Conf. Machine Learning*, 1996, pp. 71–77.
[22] P. M. Hall, D. R. Marshall, and R. Martin, "Online eigenanalysis for classification," in *Proc. BMVC*, 1998, pp. 286–295.
[23] P. M. Hall, D. R. Marshall, and R. Martin, "Merging and splitting eigenspace models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 9, pp. 1042–1049, Sep. 2000.
[24] P. M. Hall, D. R. Marshall, and R. Martin, "Adding and subtracting eigenspaces with eigenvalue decomposition and singular value decomposition," *Image Vis. Comput.*, vol. 20, no. 13/14, pp. 1009–1016, 2002.
[25] L. Wolf and A. Shashua, "Learning over sets using kernel principal angles," *J. Mach. Learn. Res.*, vol. 4, pp. 913–931, 2003.
[26] A. Georghiades, P. Belhumeur, and D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 6, pp. 643–660, Jun. 2001.
[27] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2004.
[28] O. Yamaguchi, K. Fukui, and K. Maeda, "Face recognition using temporal image sequence," in *Proc. IEEE Conf. Automatic Face and Gesture Recognition*, 1998, pp. 318–323.
[29] M. Pavlou and N. M. Allinson, "An unsupervised multiresolution face-kernel recognition model," presented at the Irish Mach. Vis. Image Processing Conf., 2005.

[30] M. Pavlou, "Face kernel extraction from local deatures," Ph.D. dissertation, Manchester Univ., Manchester, U.K., 2005.
[31] J. Ho, K.-C. Lee, M.-H. Yang, and D. Kriegman, "Visual tracking using learned linear subspaces," presented at the Computer Vision Pattern Recognition Conf., 2004.
[32] J. T. Kwok and I. W. Tsang, "The pre-image problem in kernel methods," presented at the Int. Conf. Machine Learning, 2003.

Tat-Jun Chin received the B.Eng. degree in mechatronics engineering from the Universiti Teknologi Malaysia in 2003 and the Ph.D. degree from the Department of Electrical and Computer Systems Engineering, Monash University, Victoria, Australia.

His research interest is the application of machine learning methods in computer vision.

David Suter (S'85–M'88–SM'00) received the B.Sc. degree in applied math and physics from the Flinders University of South Australia in 1977 and the Ph.D. degree in computer vision from La Trobe University in 1991.

His main research interest is motion estimation from images and visual reconstruction. He is a Professor in the Department of Electrical and Computer Systems Engineering, Monash University, Victoria, Australia. He served as General Co-Chair for several conferences. He currently serves on the editorial board of two international journals: *Machine Vision and Applications* and the *International Journal of Computer Vision*. He has previously been on the editorial board of the *International Journal of Image and Graphics*.