Eigenfaces for Recognition

Matthew Turk and Alex Pentland
Vision and Modeling Group
The Media Laboratory
Massachusetts Institute of Technology

Abstract

■ We have developed a near-real-time computer system that can locate and track a subject's head, and then recognize the person by comparing characteristics of the face to those of known individuals. The computational approach taken in this system is motivated by both physiology and information theory, as well as by the practical requirements of near-real-time performance and accuracy. Our approach treats the face recognition problem as an intrinsically two-dimensional (2-D) recognition problem rather than requiring recovery of three-dimensional geometry, taking advantage of the fact that faces are normally upright and thus may be described by a small set of 2-D characteristic views. The system functions by projecting face images onto a feature space that spans the significant variations among known face images. The significant features are known as "eigenfaces," because they are the eigenvectors (principal components) of the set of faces; they do not necessarily correspond to features such as eyes, ears, and noses. The projection operation characterizes an individual face by a weighted sum of the eigenface features, and so to recognize a particular face it is necessary only to compare these weights to those of known individuals. Some particular advantages of our approach are that it provides for the ability to learn and later recognize new faces in an unsupervised manner, and that it is easy to implement using a neural network architecture. ■

INTRODUCTION

The face is our primary focus of attention in social intercourse, playing a major role in conveying identity and emotion. Although the ability to infer intelligence or character from facial appearance is suspect, the human ability to recognize faces is remarkable. We can recognize thousands of faces learned throughout our lifetime and identify familiar faces at a glance even after years of separation. This skill is quite robust, despite large changes in the visual stimulus due to viewing conditions, expression, aging, and distractions such as glasses or changes in hairstyle or facial hair. As a consequence the visual processing of human faces has fascinated philosophers and scientists for centuries, including figures such as Aristotle and Darwin.

Computational models of face recognition, in particular, are interesting because they can contribute not only to theoretical insights but also to practical applications. Computers that recognize faces could be applied to a wide variety of problems, including criminal identification, security systems, image and film processing, and human–computer interaction. For example, the ability to model a particular face and distinguish it from a large number of stored face models would make it possible to vastly improve criminal identification. Even the ability to merely detect faces, as opposed to recognizing them, can be important. Detecting faces in photographs, for instance, is an important problem in automating color film development, since the effect of many enhancement and noise reduction techniques depends on the picture content (e.g., faces should not be tinted green, while perhaps grass should).

Unfortunately, developing a computational model of face recognition is quite difficult, because faces are complex, multidimensional, and meaningful visual stimuli. They are a natural class of objects, and stand in stark contrast to sine wave gratings, the "blocks world," and other artificial stimuli used in human and computer vision research (Davies, Ellis, & Shepherd, 1981). Thus unlike most early visual functions, for which we may construct detailed models of retinal or striate activity, face recognition is a very high level task for which computational approaches can currently only suggest broad constraints on the corresponding neural activity.

We therefore focused our research toward developing a sort of early, preattentive pattern recognition capability that does not depend on having three-dimensional information or detailed geometry. Our goal, which we believe we have reached, was to develop a computational model of face recognition that is fast, reasonably simple, and accurate in constrained environments such as an office or a household. In addition the approach is biologically implementable and is in concert with prelimi-

nary findings in the physiology and psychology of face recognition.

The scheme is based on an information theory approach that decomposes face images into a small set of characteristic feature images called "eigenfaces," which may be thought of as the principal components of the initial training set of face images. Recognition is performed by projecting a new image into the subspace spanned by the eigenfaces ("face space") and then classifying the face by comparing its position in face space with the positions of known individuals.

Automatically learning and later recognizing new faces is practical within this framework. Recognition under widely varying conditions is achieved by training on a limited number of characteristic views (e.g., a "straight on" view, a 45° view, and a profile view). The approach has advantages over other face recognition schemes in its speed and simplicity, learning capacity, and insensitivity to small or gradual changes in the face image.

Background and Related Work

Much of the work in computer recognition of faces has focused on detecting individual features such as the eyes, nose, mouth, and head outline, and defining a face model by the position, size, and relationships among these features. Such approaches have proven difficult to extend to multiple views, and have often been quite fragile, requiring a good initial guess to guide them. Research in human strategies of face recognition, moreover, has shown that individual features and their immediate relationships comprise an insufficient representation to account for the performance of adult human face identification (Carey & Diamond, 1977). Nonetheless, this approach to face recognition remains the most popular one in the computer vision literature.

Bledsoe (1966a,b) was the first to attempt semiautomated face recognition with a hybrid human–computer system that classified faces on the basis of fiducial marks entered on photographs by hand. Parameters for the classification were normalized distances and ratios among points such as eye corners, mouth corners, nose tip, and chin point. Later work at Bell Labs (Goldstein, Harmon, & Lesk, 1971; Harmon, 1971) developed a vector of up to 21 features, and recognized faces using standard pattern classification techniques. The chosen features were largely subjective evaluations (e.g., shade of hair, length of ears, lip thickness) made by human subjects, each of which would be quite difficult to automate.

An early paper by Fischler and Elschlager (1973) attempted to measure similar features automatically. They described a linear embedding algorithm that used local feature template matching and a global measure of fit to find and measure facial features. This template matching approach has been continued and improved by the recent work of Yuille, Cohen, and Hallinan (1989) (see Yuille, this volume). Their strategy is based on "deformable templates," which are parameterized models of the face and its features in which the parameter values are determined by interactions with the image.

Connectionist approaches to face identification seek to capture the configurational, or gestalt-like nature of the task. Kohonen (1989) and Kohonen and Lahtio (1981) describe an associative network with a simple learning algorithm that can recognize (classify) face images and recall a face image from an incomplete or noisy version input to the network. Fleming and Cottrell (1990) extend these ideas using nonlinear units, training the system by backpropagation. Stonham's WISARD system (1986) is a general-purpose pattern recognition device based on neural net principles. It has been applied with some success to binary face images, recognizing both identity and expression. Most connectionist systems dealing with faces (see also Midorikawa, 1988; O'Toole, Millward, & Anderson, 1988) treat the input image as a general 2-D pattern, and can make no explicit use of the configurational properties of a face. Moreover, some of these systems require an inordinate number of training examples to achieve a reasonable level of performance. Only very simple systems have been explored to date, and it is unclear how they will scale to larger problems.

Others have approached automated face recognition by characterizing a face by a set of geometric parameters and performing pattern recognition based on the parameters (e.g., Kaya & Kobayashi, 1972; Cannon, Jones, Campbell, & Morgan, 1986; Craw, Ellis, & Lishman, 1987; Wong, Law, & Tsaug, 1989). Kanade's (1973) face identification system was the first (and still one of the few) systems in which all steps of the recognition process were automated, using a top-down control strategy directed by a generic model of expected feature characteristics. His system calculated a set of facial parameters from a single face image and used a pattern classification technique to match the face from a known set, a purely statistical approach depending primarily on local histogram analysis and absolute gray-scale values.

Recent work by Burt (1988a,b) uses a "smart sensing" approach based on multiresolution template matching. This coarse-to-fine strategy uses a special-purpose computer built to calculate multiresolution pyramid images quickly, and has been demonstrated identifying people in near-real-time. This system works well under limited circumstances, but should suffer from the typical problems of correlation-based matching, including sensitivity to image size and noise. The face models are built by hand from face images.

THE EIGENFACE APPROACH

Much of the previous work on automated face recognition has ignored the issue of just what aspects of the face stimulus are important for identification. This suggested to us that an information theory approach of coding and

decoding face images may give insight into the information content of face images, emphasizing the significant local and global "features." Such features may or may not be directly related to our intuitive notion of face features such as the eyes, nose, lips, and hair. This may have important implications for the use of identification tools such as Identikit and Photofit (Bruce, 1988).

In the language of information theory, we want to extract the relevant information in a face image, encode it as efficiently as possible, and compare one face encoding with a database of models encoded similarly. A simple approach to extracting the information contained in an image of a face is to somehow capture the variation in a collection of face images, independent of any judgment of features, and use this information to encode and compare individual face images.

In mathematical terms, we wish to find the principal components of the distribution of faces, or the eigenvectors of the covariance matrix of the set of face images, treating an image as a point (or vector) in a very high dimensional space. The eigenvectors are ordered, each one accounting for a different amount of the variation among the face images.

These eigenvectors can be thought of as a set of features that together characterize the variation between face images. Each image location contributes more or less to each eigenvector, so that we can display the eigenvector as a sort of ghostly face which we call an *eigenface*. Some of the faces we studied are illustrated in Figure 1, and the corresponding eigenfaces are shown in Figure 2. Each eigenface deviates from uniform gray where some facial feature differs among the set of training faces; they are a sort of map of the variations between faces.

Each individual face can be represented exactly in terms of a linear combination of the eigenfaces. Each face can also be approximated using only the "best" eigenfaces—those that have the largest eigenvalues, and which therefore account for the most variance within the set of face images. The best M eigenfaces span an M-dimensional subspace—"face space"—of all possible images.

The idea of using eigenfaces was motivated by a technique developed by Sirovich and Kirby (1987) and Kirby and Sirovich (1990) for efficiently representing pictures of faces using principal component analysis. Starting with an ensemble of original face images, they calculated a best coordinate system for image compression, where each coordinate is actually an image that they termed an *eigenpicture*. They argued that, at least in principle, any collection of face images can be approximately reconstructed by storing a small collection of weights for each face and a small set of standard pictures (the eigenpictures). The weights describing each face are found by projecting the face image onto each eigenpicture.

It occurred to us that if a multitude of face images can be reconstructed by weighted sums of a small collection of characteristic features or eigenpictures, perhaps an efficient way to learn and recognize faces would be to build up the characteristic features by experience over time and recognize particular faces by comparing the feature weights needed to (approximately) reconstruct them with the weights associated with known individuals. Each individual, therefore, would be characterized by the small set of feature or eigenpicture weights needed to describe and reconstruct them—an extremely compact representation when compared with the images themselves.

This approach to face recognition involves the following initialization operations:

1. Acquire an initial set of face images (the training set).
2. Calculate the eigenfaces from the training set, keeping only the M images that correspond to the highest eigenvalues. These M images define the *face space*. As new faces are experienced, the eigenfaces can be updated or recalculated.
3. Calculate the corresponding distribution in M-dimensional weight space for each known individual, by projecting their face images onto the "face space."

These operations can also be performed from time to time whenever there is free excess computational capacity.

Having initialized the system, the following steps are then used to recognize new face images:

1. Calculate a set of weights based on the input image and the M eigenfaces by projecting the input image onto each of the eigenfaces.
2. Determine if the image is a face at all (whether known or unknown) by checking to see if the image is sufficiently close to "face space."
3. If it is a face, classify the weight pattern as either a known person or as unknown.
4. (Optional) Update the eigenfaces and/or weight patterns.
5. (Optional) If the same unknown face is seen several times, calculate its characteristic weight pattern and incorporate into the known faces.

Calculating Eigenfaces

Let a face image $I(x,y)$ be a two-dimensional N by N array of (8-bit) intensity values. An image may also be considered as a vector of dimension N^2, so that a typical image of size 256 by 256 becomes a vector of dimension 65,536, or, equivalently, a point in 65,536-dimensional space. An ensemble of images, then, maps to a collection of points in this huge space.

Images of faces, being similar in overall configuration, will not be randomly distributed in this huge image space and thus can be described by a relatively low dimensional subspace. The main idea of the principal compo-

Figure 1. (a)Face images used as the training set.

nent analysis (or Karhunen–Loeve expansion) is to find the vectors that best account for the distribution of face images within the entire image space. These vectors define the subspace of face images, which we call "face space." Each vector is of length N^2, describes an N by N image, and is a linear combination of the original face images. Because these vectors are the eigenvectors of the covariance matrix corresponding to the original face images, and because they are face-like in appearance, we refer to them as "eigenfaces." Some examples of eigenfaces are shown in Figure 2.

Let the training set of face images be $\Gamma_1, \Gamma_2, \Gamma_3, \ldots, \Gamma_M$. The average face of the set is defined by $\Psi = \frac{1}{M}\sum_{n=1}^{M} \Gamma_n$. Each face differs from the average by the vector $\Phi_i = \Gamma_i - \Psi$. An example training set is shown in Figure 1a, with the average face Ψ shown in Figure 1b. This set of very large vectors is then subject to principal component analysis, which seeks a set of M orthonormal vectors, $\mathbf{u}_n$, which best describes the distribution of the data. The kth vector, $\mathbf{u}_k$, is chosen such that

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M} (\mathbf{u}_k^T \Phi_n)^2 \qquad (1)$$

is a maximum, subject to

$$\mathbf{u}_l^T \mathbf{u}_k = \delta_{lk} = \begin{cases} 1, & \text{if } l = k \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

The vectors $\mathbf{u}_k$ and scalars λ_k are the eigenvectors and eigenvalues, respectively, of the covariance matrix

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T \qquad (3)$$

$$= AA^T$$

where the matrix $A = [\Phi_1\ \Phi_2 \ldots \Phi_M]$. The matrix C, however, is N^2 by N^2, and determining the N^2 eigenvectors and eigenvalues is an intractable task for typical image sizes. We need a computationally feasible method to find these eigenvectors.

If the number of data points in the image space is less than the dimension of the space ($M < N^2$), there will be only $M - 1$, rather than N^2, meaningful eigenvectors. (The remaining eigenvectors will have associated eigenvalues of zero.) Fortunately we can solve for the N^2-dimensional eigenvectors in this case by first solving for the eigenvectors of an M by M matrix—e.g., solving a 16 × 16 matrix rather than a 16,384 × 16,384 matrix—

Figure 1. (b) The average face Ψ.

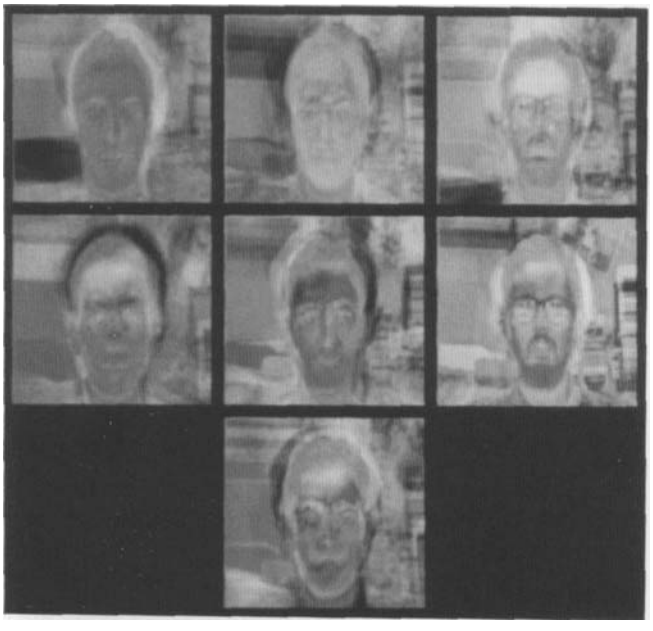


Figure 2. Seven of the eigenfaces calculated from the input images of Figure 1.

and then taking appropriate linear combinations of the face images $\mathbf{\Phi}_i$. Consider the eigenvectors $\mathbf{v}_i$ of A^TA such that

$$A^TA\mathbf{v}_i = \mu_i\mathbf{v}_i \tag{4}$$

Premultiplying both sides by A, we have

$$AA^TA\mathbf{v}_i = \mu_iA\mathbf{v}_i \tag{5}$$

from which we see that $A\mathbf{v}_i$ are the eigenvectors of $C = AA^T$.

Following this analysis, we construct the M by M matrix $L = A^TA$, where $L_{mn} = \mathbf{\Phi}_m^T\mathbf{\Phi}_n$, and find the M eigenvectors, $\mathbf{v}_l$, of L. These vectors determine linear combinations of the M training set face images to form the eigenfaces $\mathbf{u}_l$.

$$\mathbf{u}_l = \sum_{k=1}^{M} \mathbf{v}_{lk}\mathbf{\Phi}_k, \qquad l = 1, \ldots, M \tag{6}$$

With this analysis the calculations are greatly reduced, from the order of the number of pixels in the images (N^2) to the order of the number of images in the training set (M). In practice, the training set of face images will be relatively small ($M \ll N^2$), and the calculations become quite manageable. The associated eigenvalues allow us to rank the eigenvectors according to their usefulness in characterizing the variation among the images. Figure 2 shows the top seven eigenfaces derived from the input images of Figure 1.

Using Eigenfaces to Classify a Face Image

The eigenface images calculated from the eigenvectors of L span a basis set with which to describe face images. Sirovich and Kirby (1987) evaluated a limited version of this framework on an ensemble of $M = 115$ images of Caucasian males, digitized in a controlled manner, and found that about 40 eigenfaces were sufficient for a very good description of the set of face images. With $M' = 40$ eigenfaces, RMS pixel-by-pixel errors in representing cropped versions of face images were about 2%.

Since the eigenfaces seem adequate for describing face images under very controlled conditions, we decided to investigate their usefulness as a tool for face identification. In practice, a smaller M' is sufficient for identification, since accurate reconstruction of the image is not a requirement. In this framework, identification becomes a pattern recognition task. The eigenfaces span an M'-dimensional subspace of the original N^2 image space. The M' significant eigenvectors of the L matrix are chosen as those with the largest associated eigenvalues. In many of our test cases, based on $M = 16$ face images, $M' = 7$ eigenfaces were used.

A new face image ($\mathbf{\Gamma}$) is transformed into its eigenface components (projected into "face space") by a simple operation,

$$\omega_k = \mathbf{u}_k^T(\mathbf{\Gamma} - \mathbf{\Psi}) \tag{7}$$

for $k = 1, \ldots, M'$. This describes a set of point-by-point image multiplications and summations, operations performed at approximately frame rate on current image processing hardware. Figure 3 shows an image and its projection into the seven-dimensional face space.

The weights form a vector $\mathbf{\Omega}^T = [\omega_1, \omega_2 \ldots \omega_{M'}]$ that describes the contribution of each eigenface in representing the input face image, treating the eigenfaces as a basis set for face images. The vector may then be used

in a standard pattern recognition algorithm to find which of a number of predefined face classes, if any, best describes the face. The simplest method for determining which face class provides the best description of an input face image is to find the face class k that minimizes the Euclidian distance

$$\epsilon_k = \|(\mathbf{\Omega} - \mathbf{\Omega}_k)\|^2 \quad (8)$$

where $\mathbf{\Omega}_k$ is a vector describing the kth face class. The face classes $\mathbf{\Omega}_i$ are calculated by averaging the results of the eigenface representation over a small number of face images (as few as one) of each individual. A face is classified as belonging to class k when the minimum ϵ_k is below some chosen threshold θ_ϵ. Otherwise the face is classified as "unknown," and optionally used to create a new face class.

Because creating the vector of weights is equivalent to projecting the original face image onto the low-dimensional face space, many images (most of them looking nothing like a face) will project onto a given pattern vector. This is not a problem for the system, however, since the distance ϵ between the image and the face space is simply the squared distance between the mean-adjusted input image $\mathbf{\Phi} = \mathbf{\Gamma} - \mathbf{\Psi}$ and $\mathbf{\Phi}_f = \sum_{i=1}^{M'} \omega_i \mathbf{u}_i$, its projection onto face space:

$$\epsilon^2 = \|\mathbf{\Phi} - \mathbf{\Phi}_f\|^2 \quad (9)$$

Thus there are four possibilities for an input image and its pattern vector: (1) near face space and near a face class, (2) near face space but not near a known face class, (3) distant from face space and near a face class, and (4) distant from face space and not near a known face class.

In the first case, an individual is recognized and identified. In the second case, an unknown individual is present. The last two cases indicate that the image is not a face image. Case three typically shows up as a false positive in most recognition systems; in our framework, however, the false recognition may be detected because of the significant distance between the image and the subspace of expected face images. Figure 4 shows some images and their projections into face space and gives a measure of distance from the face space for each.

Summary of Eigenface Recognition Procedure

To summarize, the eigenfaces approach to face recognition involves the following steps:

1. Collect a set of characteristic face images of the known individuals. This set should include a number of images for each person, with some variation in expression and in the lighting. (Say four images of ten people, so $M = 40$.)

2. Calculate the (40×40) matrix L, find its eigenvectors and eigenvalues, and choose the M' eigenvectors with the highest associated eigenvalues. (Let $M' = 10$ in this example.)

3. Combine the normalized training set of images according to Eq. (6) to produce the ($M' = 10$) eigenfaces $\mathbf{u}_k$.

4. For each known individual, calculate the class vector $\mathbf{\Omega}_k$ by averaging the eigenface pattern vectors $\mathbf{\Omega}$ [from Eq. (8)] calculated from the original (four) images of the individual. Choose a threshold θ_ϵ that defines the maximum allowable distance from any face class, and a threshold θ_ϵ that defines the maximum allowable distance from face space [according to Eq. (9)].

5. For each new face image to be identified, calculate its pattern vector $\mathbf{\Omega}$, the distances ϵ_i to each known class, and the distance ϵ to face space. If the minimum distance $\epsilon_k < \theta_\epsilon$ and the distance $\epsilon < \theta_\epsilon$, classify the input face as the individual associated with class vector $\mathbf{\Omega}_k$. If the minimum distance $\epsilon_k > \theta_\epsilon$ but distance $\epsilon < \theta_\epsilon$, then the image may be classifed as "unknown," and optionally used to begin a new face class.

6. If the new image is classified as a known individual, this image may be added to the original set of familiar face images, and the eigenfaces may be recalculated (steps 1–4). This gives the opportunity to modify the face space as the system encounters more instances of known faces.

In our current system calculation of the eigenfaces is done offline as part of the training. The recognition currently takes about 400 msec running rather inefficiently in Lisp on a Sun4, using face images of size 128×128. With some special-purpose hardware, the current version could run at close to frame rate (33 msec).

Designing a practical system for face recognition within this framework requires assessing the tradeoffs between generality, required accuracy, and speed. If the face recognition task is restricted to a small set of people (such as the members of a family or a small company), a small set of eigenfaces is adequate to span the faces of interest. If the system is to learn new faces or represent many people, a larger basis set of eigenfaces will be required. The results of Sirovich and Kirby (1987) and Kirby and Sirovich (1990) for coding of face images gives some evidence that even if it were necessary to represent a large segment of the population, the number of eigenfaces needed would still be relatively small.

Locating and Detecting Faces

The analysis in the preceding sections assumes we have a centered face image, the same size as the training images and the eigenfaces. We need some way, then, to locate a face in a scene to do the recognition. We have developed two schemes to locate and/or track faces, using motion detection and manipulation of the images in "face space".

Figure 3. An original face image and its projection onto the face space defined by the eigenfaces of Figure 2.

Motion Detecting and Head Tracking

People are constantly moving. Even while sitting, we fidget and adjust our body position, nod our heads, look around, and such. In the case of a single person moving in a static environment, a simple motion detection and tracking algorithm, depicted in Figure 5, will locate and track the position of the head. Simple spatiotemporal filtering (e.g., frame differencing) accentuates image locations that change with time, so a moving person "lights up" in the filtered image. If the image "lights up" at all, motion is detected and the presence of a person is postulated.

After thresholding the filtered image to produce a binary motion image, we analyze the "motion blobs" over time to decide if the motion is caused by a person moving and to determine head position. A few simple rules are applied, such as "the head is the small upper blob above a larger blob (the body)," and "head motion must be reasonably slow and contiguous" (heads are not expected to jump around the image erratically). Figure 6 shows an image with the head located, along with the path of the head in the preceding sequence of frames.

The motion image also allows for an estimate of scale. The size of the blob that is assumed to be the moving head determines the size of the subimage to send to the recognition stage. This subimage is rescaled to fit the dimensions of the eigenfaces.

Using "Face Space" to Locate the Face

We can also use knowledge of the face space to locate faces in single images, either as an alternative to locating faces from motion (e.g., if there is too little motion or many moving objects) or as a method of achieving more precision than is possible by use of motion tracking alone. This method allows us to recognize the presence of faces apart from the task of identifying them.

As seen in Figure 4, images of faces do not change radically when projected into the face space, while the projection of nonface images appears quite different. This basic idea is used to detect the presence of faces in a scene: at every location in the image, calculate the distance ϵ between the local subimage and face space. This distance from face space is used as a measure of "faceness," so the result of calculating the distance from face space at every point in the image is a "face map" $\epsilon(x,y)$. Figure 7 shows an image and its face map—low values (the dark area) indicate the presence of a face.

Unfortunately, direct application of Eq. (9) is rather expensive. We have therefore developed a simpler, more efficient method of calculating the face map $\epsilon(x,y)$, which is described as follows.

To calculate the face map at every pixel of an image $I(x,y)$, we need to project the subimage centered at that pixel onto face space, then subtract the projection from the original. To project a subimage $\mathbf{\Gamma}$ onto face space, we must first subtract the mean image, resulting in $\mathbf{\Phi} = \mathbf{\Gamma} - \mathbf{\Psi}$. With $\mathbf{\Phi}_f$ being the projection of $\mathbf{\Phi}$ onto face space, the distance measure at a given image location is then

$$\begin{aligned}\epsilon^2 &= \|\mathbf{\Phi} - \mathbf{\Phi}_f\|^2 \\ &= (\mathbf{\Phi} - \mathbf{\Phi}_f)^T(\mathbf{\Phi} - \mathbf{\Phi}_f) \\ &= \mathbf{\Phi}^T\mathbf{\Phi} - \mathbf{\Phi}^T\mathbf{\Phi}_f + \mathbf{\Phi}_f^T(\mathbf{\Phi} - \mathbf{\Phi}_f) \\ &= \mathbf{\Phi}^T\mathbf{\Phi} - \mathbf{\Phi}_f^T\mathbf{\Phi}_f\end{aligned} \tag{10}$$

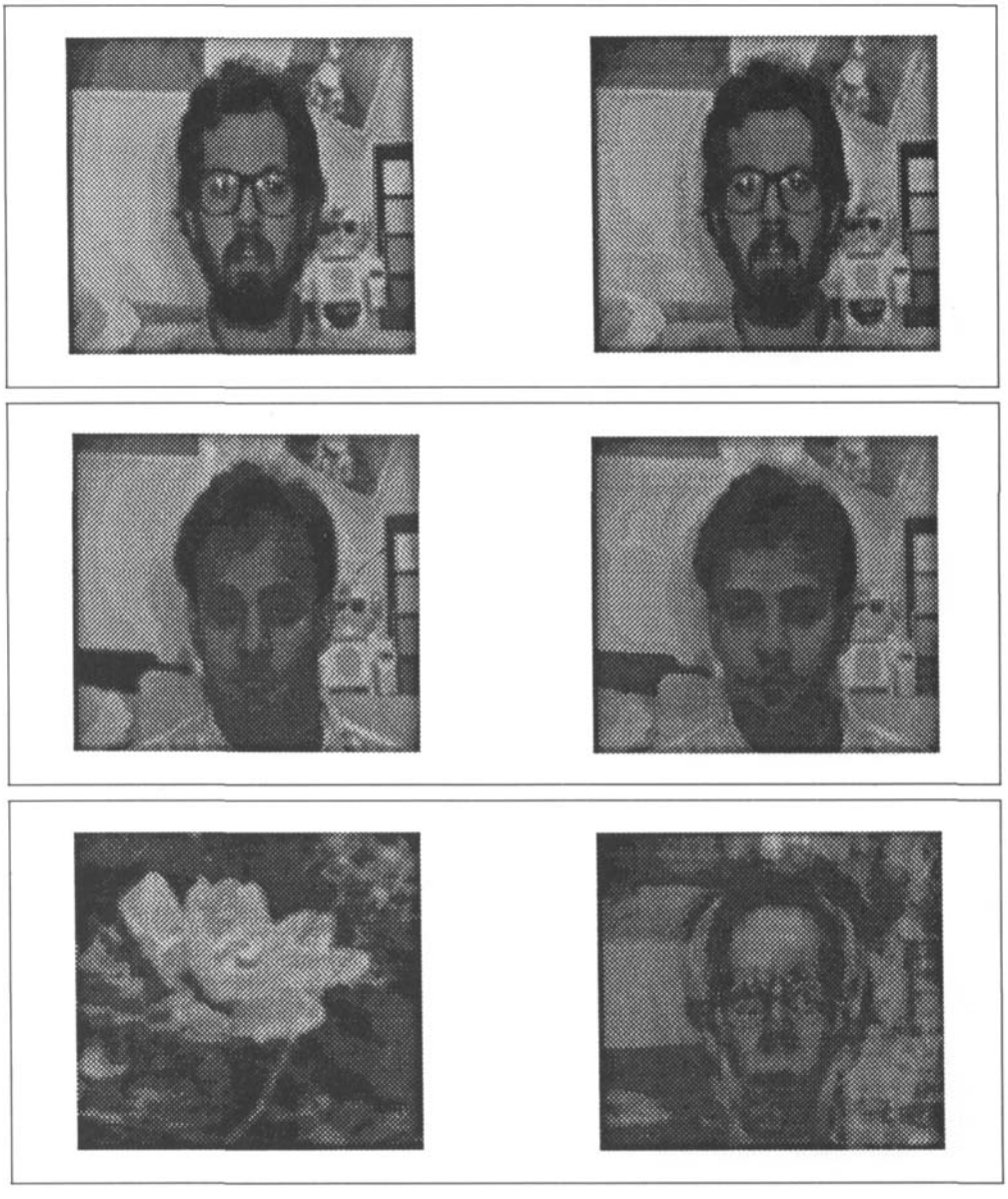


Figure 4. Three images and their projections onto the face space defined by the eigenfaces of Figure 2. The relative measures of distance from face space are **(a)** 29.8, **(b)** 58.5, **(c)** 5217.4. Images **(a)** and **(b)** are in the original training set.

since $\mathbf{\Phi}_f \perp (\mathbf{\Phi} - \mathbf{\Phi}_f)$. Because $\mathbf{\Phi}_f$ is a linear combination of the eigenfaces ($\mathbf{\Phi}_f = \sum_{i=1}^{L} \omega_i \mathbf{u}_i$) and the eigenfaces are orthonormal vectors,

$$\mathbf{\Phi}_f^T \mathbf{\Phi}_f = \sum_{i=1}^{L} \omega_i^2 \tag{11}$$

and

$$\epsilon^2(x, y) = \mathbf{\Phi}^T(x, y)\, \mathbf{\Phi}(x, y) - \sum_{i=1}^{L} \omega_i^2(x, y) \tag{12}$$

where $\epsilon(x, y)$ and $\omega_i(x, y)$ are scalar functions of image location, and $\mathbf{\Phi}(x, y)$ is a vector function of image location.

The second term of Eq. (12) is calculated in practice by a correlation with the L eigenfaces:

$$\begin{aligned} \textstyle\sum_{i=1}^{L} \omega_i^2(x, y) &= \textstyle\sum_{i=1}^{L} \mathbf{\Phi}^T(x, y)\mathbf{u}_i \\ &= \textstyle\sum_{i=1}^{L} [\mathbf{\Gamma}(x, y) - \mathbf{\Psi}]^T \mathbf{u}_i \\ &= \textstyle\sum_{i=1}^{L} [\mathbf{\Gamma}^T(x, y)\mathbf{u}_i - \mathbf{\Psi}^T \mathbf{u}_i] \\ &= \textstyle\sum_{i=1}^{L} [I(x, y) \otimes \mathbf{u}_i - \mathbf{\Psi}^T \mathbf{u}_i] \end{aligned} \tag{13}$$

where $\otimes$ is the correlation operator. The first term of Eq. (12) becomes

$$\begin{aligned} \mathbf{\Phi}^T(x, y)\mathbf{\Phi}(x, y) &= [\mathbf{\Gamma}(x, y) - \mathbf{\Psi}]^T[\mathbf{\Gamma}(x, y) - \mathbf{\Psi}] \\ &= \mathbf{\Gamma}^T(x, y)\mathbf{\Gamma}(x, y) - 2\mathbf{\Psi}^T\mathbf{\Gamma}(x, y) + \mathbf{\Psi}^T\mathbf{\Psi} \\ &= \mathbf{\Gamma}^T(x, y)\mathbf{\Gamma}(x, y) - 2\mathbf{\Gamma}(x, y) \otimes \mathbf{\Psi} + \\ &\quad \mathbf{\Psi}^T\mathbf{\Psi} \end{aligned} \tag{14}$$

so that

$$\epsilon^2(x, y) = \mathbf{\Gamma}^T(x, y)\mathbf{\Gamma}(x, y) - 2\mathbf{\Gamma}(x, y) \otimes \mathbf{\Psi} + \mathbf{\Psi}^T\mathbf{\Psi} + \sum_{i=1}^{L} [\mathbf{\Gamma}(x, y) \otimes \mathbf{u}_i - \mathbf{\Psi} \otimes \mathbf{u}_i] \tag{15}$$

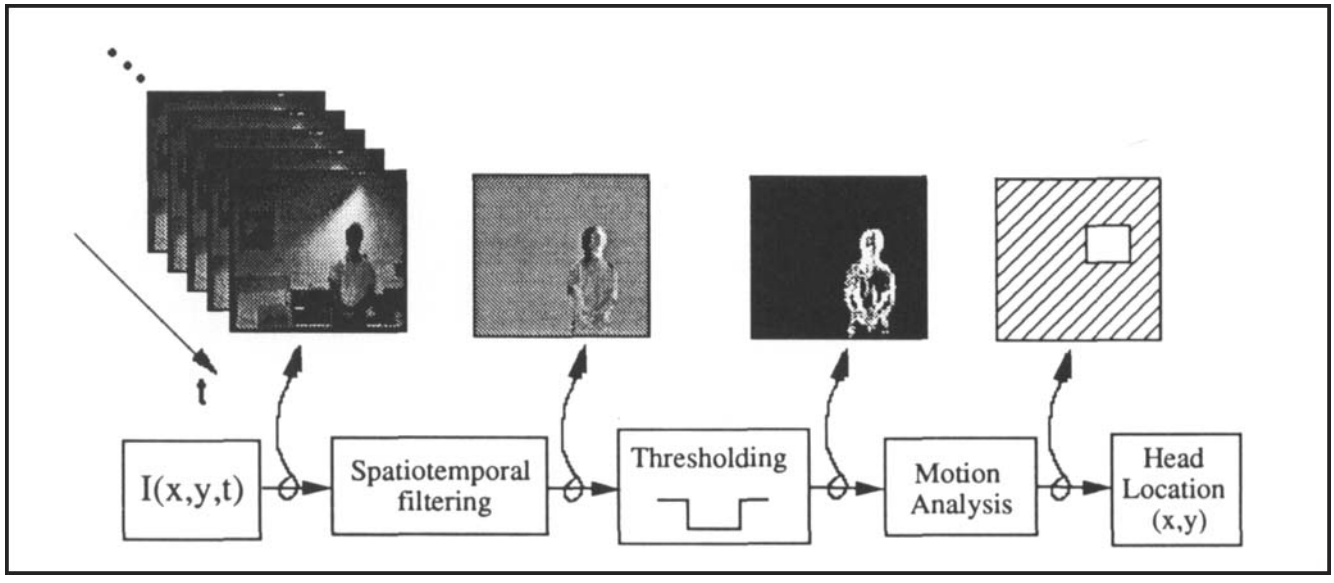


Figure 5. The head tracking and locating system.

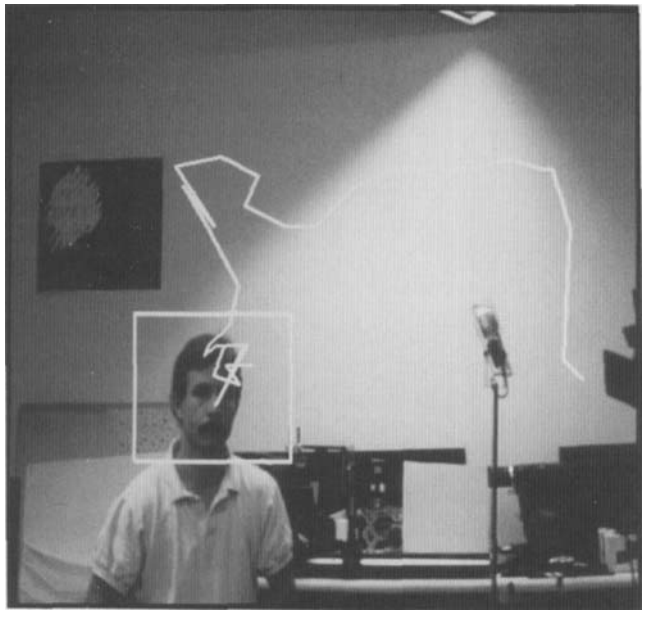


Figure 6. The head has been located—the image in the box is sent to the face recognition process. Also shown is the path of the head tracked over several previous frames.

Since the average face Ψ and the eigenfaces $\mathbf{u}_i$ are fixed, the terms $\Psi^T\Psi$ and $\Psi \otimes \mathbf{u}_i$ may be computed ahead of time.

Thus the computation of the face map involves only $L + 1$ correlations over the input image and the computation of the first term $\Gamma^T(x, y)\Gamma(x, y)$. This is computed by squaring the input image $I(x, y)$ and, at each image location, summing the squared values of the local subimage. As discussed in the section on Neural Networks, these computations can be implemented by a simple neural network.

Learning to Recognize New Faces

The concept of face space allows the ability to learn and subsequently recognize new faces in an unsupervised manner. When an image is sufficiently close to face space but is not classified as one of the familiar faces, it is initially labeled as "unknown." The computer stores the pattern vector and the corresponding unknown image. If a collection of "unknown" pattern vectors cluster in the pattern space, the presence of a new but unidentified face is postulated.

The images corresponding to the pattern vectors in the cluster are then checked for similarity by requiring that the distance from each image to the mean of the images is less than a predefined threshold. If the images pass the similarity test, the average of the feature vectors is added to the database of known faces. Occasionally, the eigenfaces may be recalculated using these stored images as part of the new training set.

Other Issues

A number of other issues must be addressed to obtain a robust working system. In this section we will briefly mention these issues and indicate methods of solution.

Eliminating the Background

In the preceding analysis we have ignored the effect of the background. In practice, the background can significantly effect the recognition performance, since the ei-

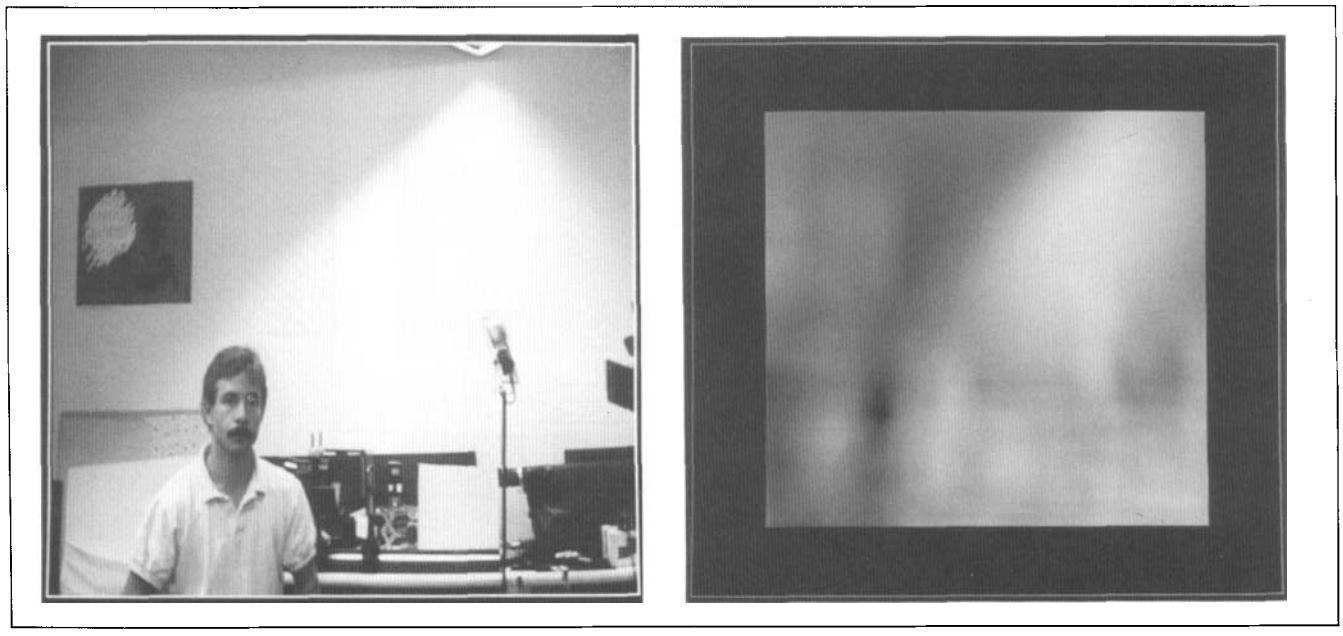


Figure 7. (a) Original image. **(b)** The corresponding face map, where low values (dark areas) indicate the presence of a face.

genface analysis as described above does not distinguish the face from the rest of the image. In the experiments described in the section on Experiments with Eigenfaces, the background was a significant part of the image used to classify the faces.

To deal with this problem without having to solve other difficult vision problems (such as robust segmentation of the head), we have multiplied the input face image by a two-dimensional gaussian window centered on the face, thus diminishing the background and accentuating the middle of the face. Experiments in human strategies of face recognition (Hay & Young, 1982) cite the importance of the internal facial features for recognition of familiar faces. Deemphasizing the outside of the face is also a practical consideration since changing hairstyles may otherwise negatively affect the recognition.

Scale (Head Size) and Orientation Invariance

The experiments in the section on Database of Face Images show that recognition performance decreases quickly as the head size, or scale, is misjudged. The head size in the input image must be close to that of the eigenfaces for the system to work well. The motion analysis gives an estimate of head size, from which the face image is rescaled to the eigenface size.

Another approach to the scale problem, which may be separate from or in addition to the motion estimate, is to use multiscale eigenfaces, in which an input face image is compared with eigenfaces at a number of scales. In this case the image will appear to be near the face space of only the closest scale eigenfaces. Equivalently, we can scale the input image to multiple sizes and use the scale that results in the smallest distance measure to face space.

Although the eigenfaces approach is not extremely sensitive to head orientation (i.e., sideways tilt of the head), a non-upright view will cause some performance degradation. An accurate estimate of the head tilt will certainly benefit the recognition. Again, two simple methods have been considered and tested. The first is to calculate the orientation of the motion blob of the head. This is less reliable as the shape tends toward a circle, however. Using the fact that faces are reasonably symmetric patterns, at least for frontal views, we have used simple symmetry operators to estimate head orientation. Once the orientation is estimated, the image can be rotated to align the head with the eigenfaces.

Distribution in Face Space

The nearest-neighbor classification previously described assumes a Gaussian distribution in face space of an individual's feature vectors $\mathbf{\Omega}$. Since there is no a priori reason to assume any particular distribution, we want to characterize it rather than assume it is gaussian. Nonlinear networks such as described in Fleming and Cottrell (1990) seem to be a promising way to learn the face space distributions by example.

Multiple Views

We are currently extending the system to deal with other than full frontal views by defining a limited number of face classes for each known person corresponding to *characteristic views*. For example, an individual may be represented by face classes corresponding to a frontal

face view, side views, at ± 45°, and right and left profile views. Under most viewing conditions these seem to be sufficient to recognize a face anywhere from frontal to profile view, because the real view can be approximated by interpolation among the fixed views.

EXPERIMENTS WITH EIGENFACES

To assess the viability of this approach to face recognition, we have performed experiments with stored face images and built a system to locate and recognize faces in a dynamic environment. We first created a large database of face images collected under a wide range of imaging conditions. Using this database we have conducted several experiments to assess the performance under known variations of lighting, scale, and orientation. The results of these experiments and early experience with the near-real-time system are reported in this section.

Database of Face Images

The images from Figure 1a were taken from a database of over 2500 face images digitized under controlled conditions. Sixteen subjects were digitized at all combinations of three head orientations, three head sizes or scales, and three lighting conditions. A six level Gaussian pyramid was constructed for each image, resulting in image resolution from 512 × 512 pixels down to 16 × 16 pixels. Figure 8 shows the images from one pyramid level for one individual.

In the first experiment the effects of varying lighting, size, and head orientation were investigated using the complete database of 2500 images of the 16 individuals shown in Figure 1a. Various groups of 16 images were selected and used as the training set. Within each training set there was one image of each person, all taken under the same conditions of lighting, image size, and head orientation. All images in the database were then classified as being one of these sixteen individuals (i.e., the threshold θ_ϵ was effectively infinite, so that no faces were rejected as unknown). Seven eigenfaces were used in the classification process.

Statistics were collected measuring the mean accuracy as a function of the difference between the training conditions and the test conditions. The independent variables were difference in illumination, imaged head size, head orientation, and combinations of illumination, size, and orientation.

Figure 9 shows results of these experiments for the case of infinite θ_ϵ. The graphs of the figure show the number of correct classifications for varying conditions of lighting, size, and head orientation, averaged over the number of experiments. For this case where every face image is classified as known, the system achieved approximately 96% correct classification averaged over lighting variation, 85% correct averaged over orientation variation, and 64% correct averaged over size variation.

As can be seen from these graphs, changing lighting conditions causes relatively few errors, while performance drops dramatically with size change. This is not surprising, since under lighting changes alone the neighborhood pixel correlation remains high, but under size changes the correlation from one image to another is largely lost. It is clear that there is a need for a multiscale approach, so that faces at a particular size are compared with one another. One method of accomplishing this is to make sure that each "face class" includes images of the individual at several different sizes, as was discussed in the section on Other Issues.

In a second experiment the same procedures were followed, but the acceptance threshold θ_ϵ was also varied. At low values of θ_ϵ, only images that project very closely to the known face classes will be recognized, so that there will be few errors but many of the images will be rejected as unknown. At high values of θ_ϵ most images will be classified, but there will be more errors. Adjusting θ_ϵ to achieve 100% accurate recognition boosted the unknown rates to 19% while varying lighting, 39% for orientation, and 60% for size. Setting the unknown rate arbitrarily to 20% resulted in correct recognition rates of 100%, 94%, and 74% respectively.

These experiments show an increase of performance accuracy as the threshold decreases. This can be tuned to achieve effectively perfect recognition as the threshold tends to zero, but at the cost of many images being rejected as unknown. The tradeoff between rejection rate and recognition accuracy will be different for each of the various face recognition applications. However, what would be most desirable is to have a way of setting the threshold high, so that few known face images are rejected as unknown, while at the same time detecting the incorrect classifications. That is, we would like to increase the efficiency (the d-prime) of the recognition process.

One way of accomplishing this is to also examine the (normalized) Euclidian distance between an image and face space as a whole. Because the projection onto the eigenface vectors is a many-to-one mapping, there is a potentially unlimited number of images that can project onto the eigenfaces in the same manner, i.e., produce the same weights. Many of these will look nothing like a face, as shown in Figure 4c. This approach was described in the section on Using "Face Space" to Locate the Face as a method of identifying likely face subimages.

Real-Time Recognition

We have used the techniques described above to build a system that locates and recognizes faces in near-real-time in a reasonably unstructured environment. Figure 10 shows a diagram of the system. A fixed camera, monitoring part of a room, is connected to a Datacube image

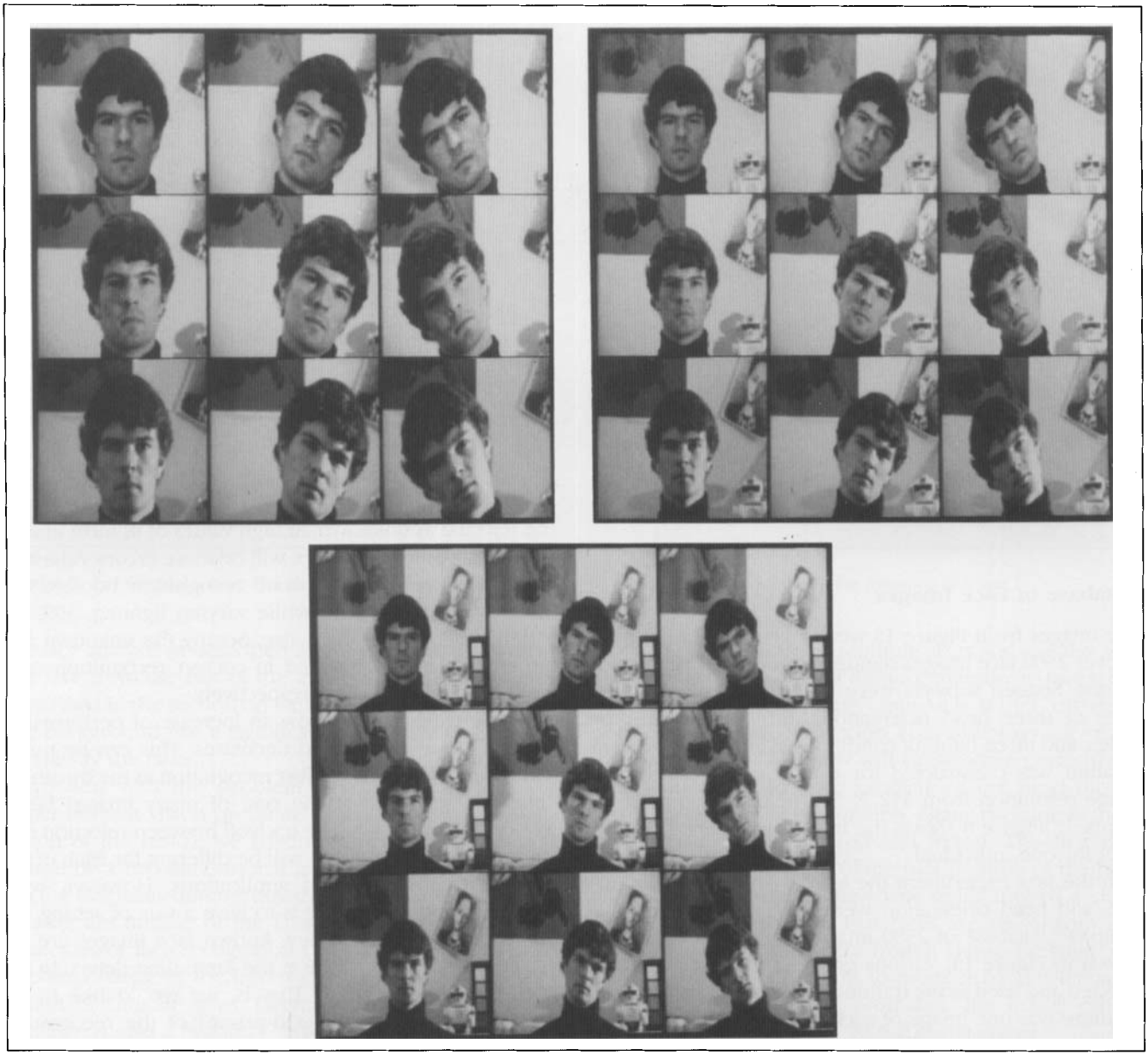


Figure 8. Variation of face images for one individual: three head sizes, three lighting conditions, and three head orientations.

processing system, which resides on the bus of a Sun 3/160. The Datacube digitizes the video image and performs spatiotemporal filtering, thresholding, and subsampling at frame rate (30 frames/sec). (The images are subsampled to speed up the motion analysis.)

The motion detection and analysis programs run on the Sun 3/160, first detecting a moving object and then tracking the motion and applying simple rules to determine if it is tracking a head. When a head is found, the subimage, centered on the head, is sent to another computer (a Sun Sparcstation) that is running the face recognition program (although it could be running on the same computer as the motion program). Using the distance-from-face-space measure, the image is either rejected as not a face, recognized as one of a group of familiar faces, or determined to be an unknown face.

Recognition occurs in this system at rates of up to two or three times per second. Until motion is detected, or as long as the image is not perceived to be a face, there is no output. When a face is recognized, the image of the identified individual is displayed on the Sun monitor.

RELATIONSHIP TO BIOLOGY AND NEURAL NETWORKS

Biological Motivations

High-level recognition tasks are typically modeled as requiring many stages of processing, e.g., the Marr (1982)

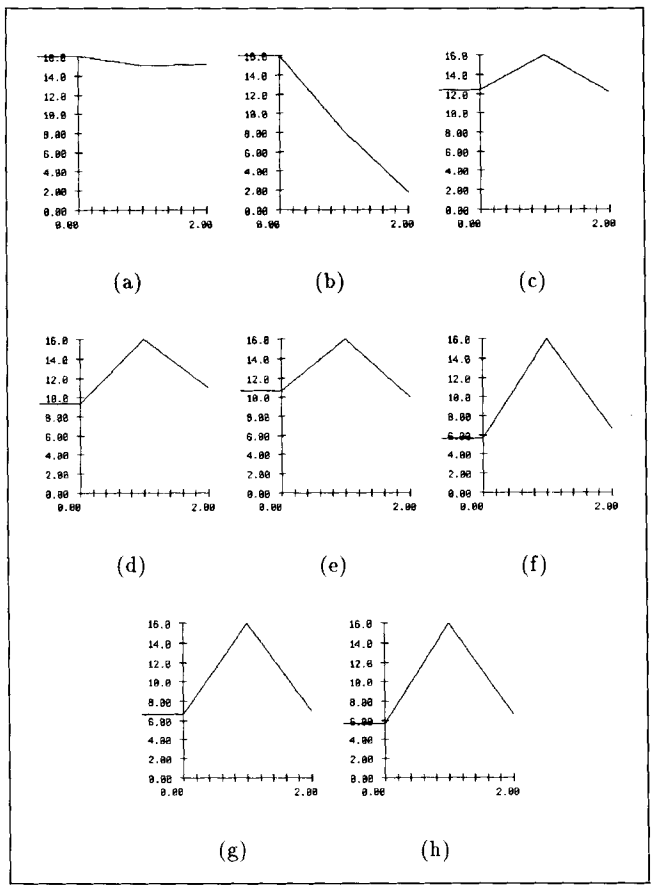


Figure 9. Results of experiments measuring recognition performance using eigenfaces. Each graph shows averaged performance as the lighting conditions, head size, and head orientation vary—the y-axis depicts number of correct classifications (out of 16). The peak (16/16 correct) in each graph results from recognizing the particular training set perfectly. The other two graph points reveal the decline in performance as the following parameters are varied: **(a)** lighting, **(b)** head size (scale), **(c)** orientation, **(d)** orientation and lighting, **(e)** orientation and size (#1), **(f)** orientation and size (#2), **(g)** size and lighting, **(h)** size and lighting (#2).

paradigm of progressing from images to surfaces to three-dimensional models to matched models. However, the early development and the extreme rapidity of face recognition makes it appear likely that there must also be a recognition mechanism based on some fast, low-level, two-dimensional image processing.

On strictly phenomenological grounds, such a face recognition mechanism is plausible because faces are typically seen in a limited range of views, and are a very important stimulus for humans from birth. The existence of such a mechanism is also supported by the results of a number of physiological experiments in monkey cortex claiming to isolate neurons that respond selectively to faces (e.g., see Perrett, Rolls, & Caan, 1982; Perrett, Mistlin, & Chitty, 1987; Bruce, Desimone, & Gross, 1981; Desimone, Albright, Gross, & Bruce, 1984; Rolls, Baylis, Hasselmo, & Nalwa, 1989). In these experiments, some cells were sensitive to identity, some to "faceness," and some only to particular views (such as frontal or profile).

Although we do not claim that biological systems have "eigenface cells" or process faces in the same way as the eigenface approach, there are a number of qualitative similarities between our approach and current understanding of human face recognition. For instance, relatively small changes cause the recognition to degrade gracefully, so that partially occluded faces can be recognized, as has been demonstrated in single-cell recording experiments. Gradual changes due to aging are easily handled by the occasional recalculation of the eigenfaces, so that the system is quite tolerant to even large changes as long as they occur over a long period of time. If, however, a large change occurs quickly—e.g., addition of a disguise or change of facial hair—then the eigenfaces approach will be fooled, as are people in conditions of casual observation.

Neural Networks

Although we have presented the eigenfaces approach to face recognition as an information-processing model, it may be implemented using simple parallel computing elements, as in a connectionist system or artificial neural network. Figure 11 shows a three-layer, fully connected linear network that implements a significant part of the system. The input layer receives the input (centered and normalized) face image, with one element per image pixel, or N elements. The weights from the input layer to the hidden layer correspond to the eigenfaces, so that the value of each hidden unit is the dot product of the input image and the corresponding eigenface: $\omega_i = \mathbf{\Phi}^T \mathbf{u}_i$. The hidden units, then, form the pattern vector $\mathbf{\Omega}^T = [\omega_1, \omega_2 \ldots \omega_L]$.

The output layer produces the face space projection of the input image when the output weights also correspond to the eigenfaces (mirroring the input weights). Adding two nonlinear components we construct Figure 12, which produces the pattern class $\mathbf{\Omega}$, face space projection $\mathbf{\Phi}_f$, distance measure d (between the image and its face space projection), and a classification vector. The classification vector is comprised of a unit for each known face defining the pattern space distances ϵ_i. The unit with the smallest value, if below the specified threshold θ_ϵ, reveals the identity of the input face image.

Parts of the network of Figure 12 are similar to the associative networks of Kohonen (1989) and Kohonen and Lehtio (1981). These networks implement a learned stimulus–response mapping, in which the learning phase modifies the connection weights. An autoassociative network implements the projection onto face space. Similarly, reconstruction using eigenfaces can be used to recall a partially occluded face, as shown in Figure 13.

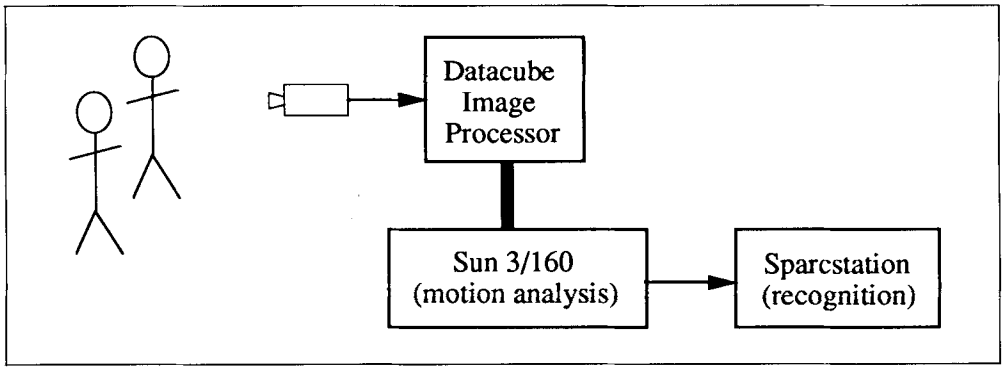


Figure 10. System diagram of the face recognition system.

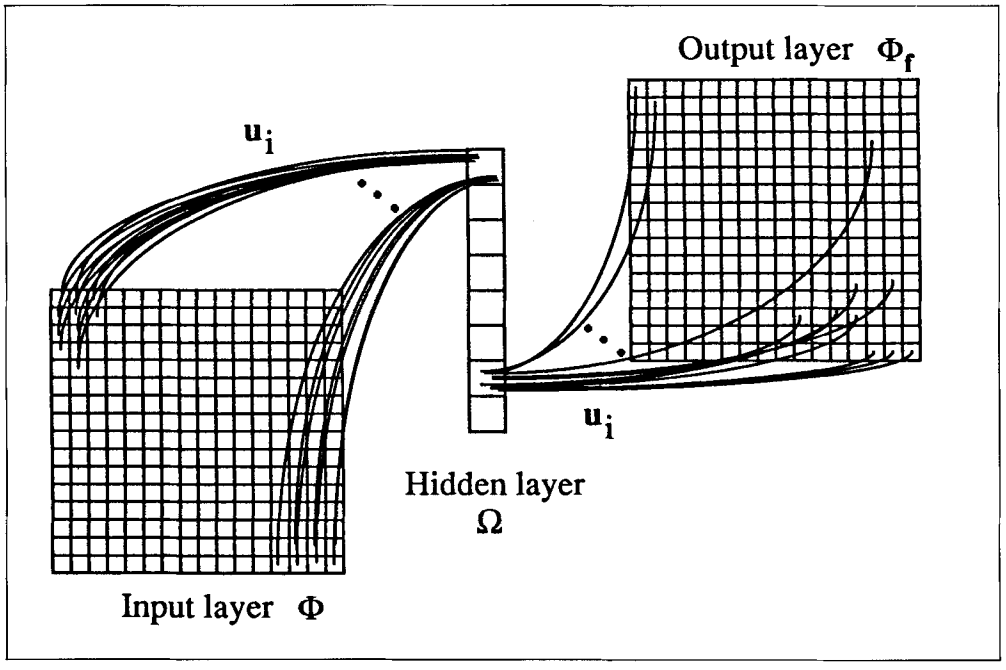


Figure 11. Three-layer linear network for eigenface calculation. The symmetric weights $\mathbf{u}_i$ are the eigenfaces, and the hidden units reveal the projection of the input image Φ onto the eigenfaces. The output Φ_f is the face space projection of the input image.

CONCLUSION

Early attempts at making computers recognize faces were limited by the use of impoverished face models and feature descriptions (e.g., locating features from an edge image and matching simple distances and ratios), assuming that a face is no more than the sum of its parts, the individual features. Recent attempts using parameterized feature models and multiscale matching look more promising, but still face severe problems before they are generally applicable. Current connectionist approaches tend to hide much of the pertinent information in the weights that makes it difficult to modify and evaluate parts of the approach.

The eigenface approach to face recognition was motivated by information theory, leading to the idea of basing face recognition on a small set of image features that best approximates the set of known face images, without requiring that they correspond to our intuitive notions of facial parts and features. Although it is not an elegant solution to the general recognition problem, the eigenface approach does provide a practical solution that is well fitted to the problem of face recognition. It is fast, relatively simple, and has been shown to work well in a constrained environment. It can also be implemented using modules of connectionist or neural networks.

It is important to note that many applications of face recognition do not require perfect identification, although most require a low false-positive rate. In searching a large database of faces, for example, it may be preferable to find a small set of likely matches to present to the user. For applications such as security systems or human–computer interaction, the system will normally be able to "view" the subject for a few seconds or minutes, and thus will have a number of chances to recognize the person. Our experiments show that the eigenface technique can be made to perform at very high accuracy, although with a substantial "unknown" rejection rate, and thus is potentially well suited to these applications.

We are currently investigating in more detail the issues of robustness to changes in lighting, head size, and head orientation, automatically learning new faces, incorpo-

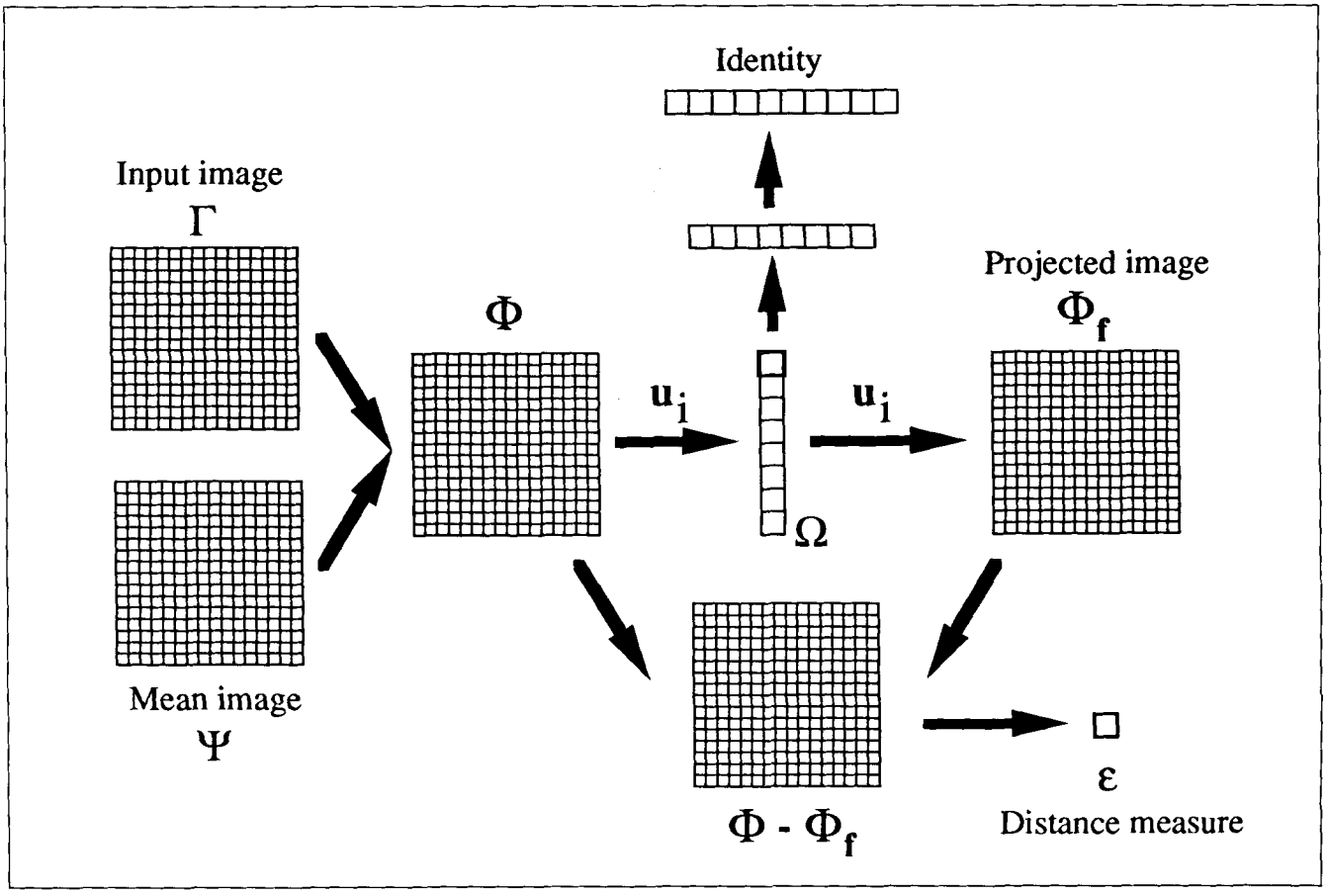


Figure 12. Collection of networks to implement computation of the pattern vector, projection into face space, distance from face space measure, and identification.

Figure 13. **(a)** Partially occluded face image and **(b)** its reconstruction using the eigenfaces.

rating a limited number of characteristic views for each individual, and the tradeoffs between the number of people the system needs to recognize and the number of eigenfaces necessary for unambiguous classification. In addition to recognizing faces, we are also beginning efforts to use eigenface analysis to determine the gender of the subject and to interpret facial expressions, two important face processing problems that complement the task of face recognition.

REFERENCES

Bledsoe, W. W. (1966a). The model method in facial recognition. Panoramic Research Inc., Palo Alto, CA, Rep. PRI:15, August.

Bledsoe, W. W. (1966b). Man-machine facial recognition. Panoramic Research Inc., Palo Alto, CA, Rep. PRI:22, August.

Bruce, V. (1988). *Recognising faces.* Hillsdale, NJ: Erlbaum.

Bruce, C. J., Desimone, R., & Gross, C. G. (1981). *Journal of Neurophysiology, 46,* 369–384.

Burt, P. (1988a). Algorithms and architectures for smart sensing. *Proceedings of the Image Understanding Workshop,* April.

Burt, P. (1988b). Smart sensing within a Pyramid Vision Machine. *Proceedings of IEEE, 76*(8), 139–153.

Cannon, S. R., Jones, G. W., Campbell, R., & Morgan, N. W. (1986). A computer vision system for identification of individuals. *Proceedings of IECON, 1.*

Carey, S., & Diamond, R. (1977). From piecemeal to configurational representation of faces. *Science, 195,* 312–313.

Craw, Ellis, & Lishman (1987). Automatic extraction of face features. *Pattern Recognition Letters, 5,* 183–187.

Davies, Ellis, & Shepherd (Eds.), (1981). *Perceiving and remembering faces.* London: Academic Press.

Desimone, R., Albright, T. D., Gross, C. G., & Bruce, C. J. (1984). Stimulus-selective properties of inferior temporal neurons in the macaque. *Neuroscience, 4,* 2051–2068.

Fischler, M. A., & Elschlager, R. A. (1973). The representation and matching of pictorial structures. *IEEE Transactions on Computers, c-22*(1).

Fleming, M., & Cottrell, G. (1990). Categorization of faces using unsupervised feature extraction. *Proceedings of IJCNN-90, 2.*

Goldstein, Harmon, & Lesk (1971). Identification of human faces. *Proceedings IEEE, 59,* 748.

Harmon, L. D. (1971). Some aspects of recognition of human faces. In O. J. Grusser & R. Klinke (Eds.), *Pattern recognition in biological and technical systems.* Berlin: Springer-Verlag.

Hay, D. C., & Young, A. W. (1982). The human face. In A. W. Ellis (Ed.), *Normality and pathology in cognitive functions.* London: Academic Press.

Kanade, T. (1973). Picture processing system by computer complex and recognition of human faces. Dept. of Information Science, Kyoto University.

Kaya, Y., & Kobayashi, K. (1972). A basic study on human face recognition. In S. Watanabe (Ed.), *Frontiers of pattern recognition.* New York: Academic Press.

Kirby, M., & Sirovich, L. (1990). Application of the Karhunen–Loeve procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12*(1).

Kohonen, T. (1989). *Self-organization and associative memory.* Berlin: Springer-Verlag.

Kohonen, T., & Lehtio, P. (1981). Storage and processing of information in distributed associative memory systems. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory.* Hillsdale, NJ: Erlbaum, pp. 105–143.

Marr, D. (1982). *Vision.* San Francisco: W. H. Freeman.

Midorikawa, H. (1988). The face pattern identification by back-propagation learning procedure. *Abstracts of the First Annual INNS Meeting,* Boston, p. 515.

O'Toole, Millward, & Anderson (1988). A physical system approach to recognition memory for spatially transformed faces. *Neural Networks, 1,* 179–199.

Perrett, D. I., Mistlin, A. J., & Chitty, A. J. (1987). Visual neurones responsive to faces. *TINS, 10*(9), 358–364.

Perrett, Rolls, & Caan (1982). Visual neurones responsive to faces in the monkey temporal cortex. *Experimental Brain Research, 47,* 329–342.

Rolls, E. T., Baylis, G. C., Hasselmo, M. E., & Nalwa, V. (1989). The effect of learning on the face selective responses of neurons in the cortex in the superior temporal sulcus of the monkey. *Experimental Brain Research, 76,* 153–164.

Sirovich, L., & Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A, 4*(3), 519–524.

Stonham, T. J. (1986). Practical face recognition and verification with WISARD. In H. Ellis, M. Jeeves, F. Newcombe, & A. Young (Eds.), *Aspects of face processing.* Dordrecht: Martinus Nijhoff.

Wong, K., Law, H., & Tsang, P. (1989). A system for recognising human faces. *Proceedings of ICASSP,* May, 1638–1642.

Yuille, A. L., Cohen, D. S., & Hallinan, P. W. (1989). Feature extraction from faces using deformable templates. *Proceedings of CVPR,* San Diego, CA, June.

A Tutorial on Principal Component Analysis

Jonathon Shlens*

Google Research
Mountain View, CA 94043

(Dated: April 7, 2014; Version 3.02)

Principal component analysis (PCA) is a mainstay of modern data analysis - a black box that is widely used but (sometimes) poorly understood. The goal of this paper is to dispel the magic behind this black box. This manuscript focuses on building a solid intuition for how and why principal component analysis works. This manuscript crystallizes this knowledge by deriving from simple intuitions, the mathematics behind PCA. This tutorial does not shy away from explaining the ideas informally, nor does it shy away from the mathematics. The hope is that by addressing both aspects, readers of all levels will be able to gain a better understanding of PCA as well as the when, the how and the why of applying this technique.

I. INTRODUCTION

Principal component analysis (PCA) is a standard tool in modern data analysis - in diverse fields from neuroscience to computer graphics - because it is a simple, non-parametric method for extracting relevant information from confusing data sets. With minimal effort PCA provides a roadmap for how to reduce a complex data set to a lower dimension to reveal the sometimes hidden, simplified structures that often underlie it.

The goal of this tutorial is to provide both an intuitive feel for PCA, and a thorough discussion of this topic. We will begin with a simple example and provide an intuitive explanation of the goal of PCA. We will continue by adding mathematical rigor to place it within the framework of linear algebra to provide an explicit solution. We will see how and why PCA is intimately related to the mathematical technique of singular value decomposition (SVD). This understanding will lead us to a prescription for how to apply PCA in the real world and an appreciation for the underlying assumptions. My hope is that a thorough understanding of PCA provides a foundation for approaching the fields of machine learning and dimensional reduction.

The discussion and explanations in this paper are informal in the spirit of a tutorial. The goal of this paper is to *educate*. Occasionally, rigorous mathematical proofs are necessary although relegated to the Appendix. Although not as vital to the tutorial, the proofs are presented for the adventurous reader who desires a more complete understanding of the math. My only assumption is that the reader has a working knowledge of linear algebra. My goal is to provide a thorough discussion by largely building on ideas from linear algebra and avoiding challenging topics in statistics and optimization theory (but see Discussion). Please feel free to contact me with any suggestions, corrections or comments.

*Electronic address: jonathon.shlens@gmail.com

II. MOTIVATION: A TOY EXAMPLE

Here is the perspective: we are an experimenter. We are trying to understand some phenomenon by measuring various quantities (e.g. spectra, voltages, velocities, etc.) in our system. Unfortunately, we can not figure out what is happening because the data appears clouded, unclear and even redundant. This is not a trivial problem, but rather a fundamental obstacle in empirical science. Examples abound from complex systems such as neuroscience, web indexing, meteorology and oceanography - the number of variables to measure can be unwieldy and at times even *deceptive*, because the underlying relationships can often be quite simple.

Take for example a simple toy problem from physics diagrammed in Figure 1. Pretend we are studying the motion of the physicist's ideal spring. This system consists of a ball of mass m attached to a massless, frictionless spring. The ball is released a small distance away from equilibrium (i.e. the spring is stretched). Because the spring is ideal, it oscillates indefinitely along the x-axis about its equilibrium at a set frequency.

This is a standard problem in physics in which the motion along the x direction is solved by an explicit function of time. In other words, the underlying dynamics can be expressed as a function of a single variable x.

However, being ignorant experimenters we do not know any of this. We do not know which, let alone how many, axes and dimensions are important to measure. Thus, we decide to measure the ball's position in a three-dimensional space (since we live in a three dimensional world). Specifically, we place three movie cameras around our system of interest. At 120 Hz each movie camera records an image indicating a two dimensional position of the ball (a projection). Unfortunately, because of our ignorance, we do not even know what are the real x, y and z axes, so we choose three camera positions $\vec{\mathbf{a}}, \vec{\mathbf{b}}$ and $\vec{\mathbf{c}}$ at some arbitrary angles with respect to the system. The angles between our measurements might not even be 90^o! Now, we record with the cameras for several minutes. The big question remains: *how do we get from this data set to a simple equation*

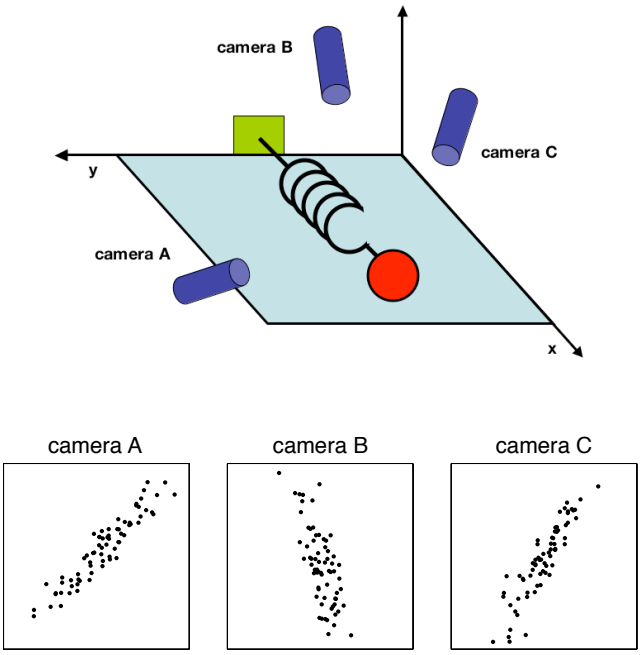


FIG. 1 A toy example. The position of a ball attached to an oscillating spring is recorded using three cameras A, B and C. The position of the ball tracked by each camera is depicted in each panel below.

of x?

We know a-priori that if we were smart experimenters, we would have just measured the position along the x-axis with one camera. But this is not what happens in the real world. We often do not know which measurements best reflect the dynamics of our system in question. Furthermore, we sometimes record more dimensions than we actually need.

Also, we have to deal with that pesky, real-world problem of noise. In the toy example this means that we need to deal with air, imperfect cameras or even friction in a less-than-ideal spring. Noise contaminates our data set only serving to obfuscate the dynamics further. *This toy example is the challenge experimenters face everyday.* Keep this example in mind as we delve further into abstract concepts. Hopefully, by the end of this paper we will have a good understanding of how to systematically extract x using principal component analysis.

III. FRAMEWORK: CHANGE OF BASIS

The goal of principal component analysis is to identify the most meaningful basis to re-express a data set. The hope is that this new basis will filter out the noise and reveal hidden structure. In the example of the spring, the explicit goal of PCA is to determine: "the dynamics are along the x-axis." In other words, the goal of PCA is to determine that $\hat{\mathbf{x}}$, i.e. the unit basis vector along the x-axis, is the important dimension. Determining this fact allows an experimenter to discern which dynamics are important, redundant or noise.

A. A Naive Basis

With a more precise definition of our goal, we need a more precise definition of our data as well. We treat every time sample (or experimental trial) as an individual sample in our data set. At each time sample we record a set of data consisting of multiple measurements (e.g. voltage, position, etc.). In our data set, at one point in time, camera A records a corresponding ball position (x_A, y_A). One sample or trial can then be expressed as a 6 dimensional column vector

$$\vec{X} = \begin{bmatrix} x_A \\ y_A \\ x_B \\ y_B \\ x_C \\ y_C \end{bmatrix}$$

where each camera contributes a 2-dimensional projection of the ball's position to the entire vector $\vec{X}$. If we record the ball's position for 10 minutes at 120 Hz, then we have recorded $10 \times 60 \times 120 = 72000$ of these vectors.

With this concrete example, let us recast this problem in abstract terms. Each sample $\vec{X}$ is an m-dimensional vector, where m is the number of measurement types. Equivalently, every sample is a vector that lies in an m-dimensional vector space spanned by some orthonormal basis. From linear algebra we know that all measurement vectors form a linear combination of this set of unit length basis vectors. What is this orthonormal basis?

This question is usually a tacit assumption often overlooked. Pretend we gathered our toy example data above, but only looked at camera A. What is an orthonormal basis for (x_A, y_A)? A naive choice would be $\{(1,0),(0,1)\}$, but why select this basis over $\{(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}),(\frac{-\sqrt{2}}{2}, \frac{-\sqrt{2}}{2})\}$ or any other arbitrary rotation? The reason is that the *naive basis reflects the method we gathered the data.* Pretend we record the position $(2,2)$. We did not record $2\sqrt{2}$ in the $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$ direction and 0 in the perpendicular direction. Rather, we recorded the position $(2,2)$ on our camera meaning 2 units up and 2 units to the left in our camera window. Thus our original basis reflects the method we measured our data.

How do we express this naive basis in linear algebra? In the two dimensional case, $\{(1,0),(0,1)\}$ can be recast as individual row vectors. A matrix constructed out of these row vectors is the 2×2 identity matrix I. We can generalize this to the m-dimensional case by constructing an $m \times m$ identity matrix

$$\mathbf{B} = \begin{bmatrix} \mathbf{b_1} \\ \mathbf{b_2} \\ \vdots \\ \mathbf{b_m} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = \mathbf{I}$$

where each *row* is an orthornormal basis vector $\mathbf{b}_i$ with m components. We can consider our naive basis as the effective starting point. All of our data has been recorded in this basis

and thus it can be trivially expressed as a linear combination of $\{\mathbf{b}_i\}$.

B. Change of Basis

With this rigor we may now state more precisely what PCA asks: *Is there another basis, which is a linear combination of the original basis, that best re-expresses our data set?*

A close reader might have noticed the conspicuous addition of the word *linear*. Indeed, PCA makes one stringent but powerful assumption: linearity. Linearity vastly simplifies the problem by restricting the set of potential bases. With this assumption PCA is now limited to re-expressing the data as a *linear combination* of its basis vectors.

Let $\mathbf{X}$ be the original data set, where each *column* is a single sample (or moment in time) of our data set (i.e. $\vec{X}$). In the toy example $\mathbf{X}$ is an $m \times n$ matrix where $m = 6$ and $n = 72000$. Let $\mathbf{Y}$ be another $m \times n$ matrix related by a linear transformation $\mathbf{P}$. $\mathbf{X}$ is the original recorded data set and $\mathbf{Y}$ is a new representation of that data set.

$$\mathbf{PX} = \mathbf{Y} \tag{1}$$

Also let us define the following quantities.[1]

- $\mathbf{p_i}$ are the rows of $\mathbf{P}$
- $\mathbf{x_i}$ are the columns of $\mathbf{X}$ (or individual $\vec{X}$).
- $\mathbf{y_i}$ are the columns of $\mathbf{Y}$.

Equation 1 represents a change of basis and thus can have many interpretations.

1. $\mathbf{P}$ is a matrix that transforms $\mathbf{X}$ into $\mathbf{Y}$.
2. Geometrically, $\mathbf{P}$ is a rotation and a stretch which again transforms $\mathbf{X}$ into $\mathbf{Y}$.
3. The rows of $\mathbf{P}$, $\{\mathbf{p_1}, \ldots, \mathbf{p_m}\}$, are a set of new basis vectors for expressing the columns of $\mathbf{X}$.

The latter interpretation is not obvious but can be seen by writing out the explicit dot products of $\mathbf{PX}$.

$$\mathbf{PX} = \begin{bmatrix} \mathbf{p_1} \\ \vdots \\ \mathbf{p_m} \end{bmatrix} \begin{bmatrix} \mathbf{x_1} & \cdots & \mathbf{x_n} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{p_1} \cdot \mathbf{x_1} & \cdots & \mathbf{p_1} \cdot \mathbf{x_n} \\ \vdots & \ddots & \vdots \\ \mathbf{p_m} \cdot \mathbf{x_1} & \cdots & \mathbf{p_m} \cdot \mathbf{x_n} \end{bmatrix}$$

[1] In this section $\mathbf{x_i}$ and $\mathbf{y_i}$ are *column* vectors, but be forewarned. In all other sections $\mathbf{x_i}$ and $\mathbf{y_i}$ are *row* vectors.

We can note the form of each column of $\mathbf{Y}$.

$$\mathbf{y}_i = \begin{bmatrix} \mathbf{p_1} \cdot \mathbf{x_i} \\ \vdots \\ \mathbf{p_m} \cdot \mathbf{x_i} \end{bmatrix}$$

We recognize that each coefficient of $\mathbf{y_i}$ is a dot-product of $\mathbf{x_i}$ with the corresponding row in $\mathbf{P}$. In other words, the j^{th} coefficient of $\mathbf{y_i}$ is a projection on to the j^{th} row of $\mathbf{P}$. This is in fact the very form of an equation where $\mathbf{y_i}$ is a projection on to the basis of $\{\mathbf{p_1}, \ldots, \mathbf{p_m}\}$. Therefore, the rows of $\mathbf{P}$ are a new set of basis vectors for representing of columns of $\mathbf{X}$.

C. Questions Remaining

By assuming linearity the problem reduces to finding the appropriate *change of basis*. The row vectors $\{\mathbf{p_1}, \ldots, \mathbf{p_m}\}$ in this transformation will become the *principal components* of $\mathbf{X}$. Several questions now arise.

- What is the best way to re-express $\mathbf{X}$?
- What is a good choice of basis $\mathbf{P}$?

These questions must be answered by next asking ourselves *what features we would like* $\mathbf{Y}$ *to exhibit*. Evidently, additional assumptions beyond linearity are required to arrive at a reasonable result. The selection of these assumptions is the subject of the next section.

IV. VARIANCE AND THE GOAL

Now comes the most important question: what does *best express* the data mean? This section will build up an intuitive answer to this question and along the way tack on additional assumptions.

A. Noise and Rotation

Measurement noise in any data set must be low or else, no matter the analysis technique, no information about a signal can be extracted. There exists no absolute scale for noise but rather all noise is quantified relative to the signal strength. A common measure is the *signal-to-noise ratio* (*SNR*), or a ratio of variances σ^2,

$$SNR = \frac{\sigma^2_{signal}}{\sigma^2_{noise}}.$$

A high SNR ($\gg 1$) indicates a high precision measurement, while a low SNR indicates very noisy data.

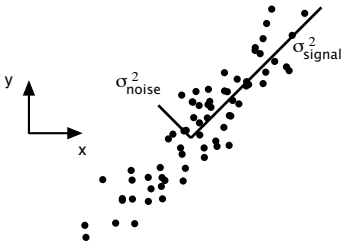


FIG. 2 Simulated data of (x,y) for camera A. The signal and noise variances σ^2_{signal} and σ^2_{noise} are graphically represented by the two lines subtending the cloud of data. Note that the largest direction of variance does not lie along the basis of the recording (x_A, y_A) but rather along the best-fit line.

FIG. 3 A spectrum of possible redundancies in data from the two separate measurements r_1 and r_2. The two measurements on the left are uncorrelated because one can not predict one from the other. Conversely, the two measurements on the right are highly correlated indicating highly redundant measurements.

Let's take a closer examination of the data from camera A in Figure 2. Remembering that the spring travels in a straight line, every individual camera should record motion in a straight line as well. Therefore, any spread deviating from straight-line motion is noise. The variance due to the signal and noise are indicated by each line in the diagram. The ratio of the two lengths measures how skinny the cloud is: possibilities include a thin line (SNR $\gg 1$), a circle (SNR $= 1$) or even worse. By positing reasonably good measurements, quantitatively we assume that directions with largest variances in our measurement space contain the dynamics of interest. In Figure 2 the direction with the largest variance is not $\hat{x}_A = (1,0)$ nor $\hat{y}_A = (0,1)$, but the direction along the long axis of the cloud. Thus, by assumption the dynamics of interest exist along directions with largest variance and presumably highest SNR.

Our assumption suggests that the basis for which we are searching is not the naive basis because these directions (i.e. (x_A, y_A)) do not correspond to the directions of largest variance. Maximizing the variance (and by assumption the SNR) corresponds to finding the appropriate rotation of the naive basis. This intuition corresponds to finding the direction indicated by the line σ^2_{signal} in Figure 2. In the 2-dimensional case of Figure 2 the direction of largest variance corresponds to the best-fit line for the data cloud. Thus, rotating the naive basis to lie parallel to the best-fit line would reveal the direction of motion of the spring for the 2-D case. How do we generalize this notion to an arbitrary number of dimensions? Before we approach this question we need to examine this issue from a second perspective.

B. Redundancy

Figure 2 hints at an additional confounding factor in our data - redundancy. This issue is particularly evident in the example of the spring. In this case multiple sensors record the same dynamic information. Reexamine Figure 2 and ask whether it was really necessary to record 2 variables. Figure 3 might reflect a range of possibile plots between two arbitrary measurement types r_1 and r_2. The left-hand panel depicts two recordings with no apparent relationship. Because one can not predict r_1 from r_2, one says that r_1 and r_2 are uncorrelated.

On the other extreme, the right-hand panel of Figure 3 depicts highly correlated recordings. This extremity might be achieved by several means:

- A plot of (x_A, x_B) if cameras A and B are very nearby.
- A plot of $(x_A, \tilde{x}_A)$ where x_A is in meters and $\tilde{x}_A$ is in inches.

Clearly in the right panel of Figure 3 it would be more meaningful to just have recorded a single variable, not both. Why? Because one can calculate r_1 from r_2 (or vice versa) using the best-fit line. Recording solely one response would express the data more concisely and reduce the number of sensor recordings ($2 \to 1$ variables). Indeed, this is the central idea behind dimensional reduction.

C. Covariance Matrix

In a 2 variable case it is simple to identify redundant cases by finding the slope of the best-fit line and judging the quality of the fit. How do we quantify and generalize these notions to arbitrarily higher dimensions? Consider two sets of measurements with zero means

$$A = \{a_1, a_2, \ldots, a_n\} \ , \quad B = \{b_1, b_2, \ldots, b_n\}$$

where the subscript denotes the sample number. The variance of A and B are individually defined as,

$$\sigma_A^2 = \frac{1}{n}\sum_i a_i^2, \quad \sigma_B^2 = \frac{1}{n}\sum_i b_i^2$$

The *covariance* between A and B is a straight-forward generalization.

$$\textit{covariance of } A \textit{ and } B \equiv \sigma_{AB}^2 = \frac{1}{n}\sum_i a_i b_i$$

The covariance measures the degree of the linear relationship between two variables. A large positive value indicates positively correlated data. Likewise, a large negative value denotes negatively correlated data. The absolute magnitude of the covariance measures the degree of redundancy. Some additional facts about the covariance.

- σ_{AB} is zero if and only if A and B are uncorrelated (e.g. Figure 2, left panel).
- $\sigma_{AB}^2 = \sigma_A^2$ if $A = B$.

We can equivalently convert A and B into corresponding row vectors.

$$\begin{aligned} \mathbf{a} &= [a_1\ a_2\ \ldots\ a_n] \\ \mathbf{b} &= [b_1\ b_2\ \ldots\ b_n] \end{aligned}$$

so that we may express the covariance as a dot product matrix computation.[2]

$$\sigma_{\mathbf{ab}}^2 \equiv \frac{1}{n}\mathbf{a}\mathbf{b}^T \tag{2}$$

Finally, we can generalize from two vectors to an arbitrary number. Rename the row vectors $\mathbf{a}$ and $\mathbf{b}$ as $\mathbf{x_1}$ and $\mathbf{x_2}$, respectively, and consider additional indexed row vectors $\mathbf{x_3}, \ldots, \mathbf{x_m}$. Define a new $m \times n$ matrix $\mathbf{X}$.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x_1} \\ \vdots \\ \mathbf{x_m} \end{bmatrix}$$

One interpretation of $\mathbf{X}$ is the following. Each *row* of $\mathbf{X}$ corresponds to all measurements of a particular type. Each *column* of $\mathbf{X}$ corresponds to a set of measurements from one particular trial (this is $\vec{X}$ from section 3.1). We now arrive at a definition for the *covariance matrix* $\mathbf{C_X}$.

$$\mathbf{C_X} \equiv \frac{1}{n}\mathbf{X}\mathbf{X}^T.$$

Consider the matrix $\mathbf{C_X} = \frac{1}{n}\mathbf{X}\mathbf{X}^T$. The ij^{th} element of $\mathbf{C_X}$ is the dot product between the vector of the i^{th} measurement type with the vector of the j^{th} measurement type. We can summarize several properties of $\mathbf{C_X}$:

- $\mathbf{C_X}$ is a square symmetric $m \times m$ matrix (Theorem 2 of Appendix A)
- The diagonal terms of $\mathbf{C_X}$ are the *variance* of particular measurement types.
- The off-diagonal terms of $\mathbf{C_X}$ are the *covariance* between measurement types.

$\mathbf{C_X}$ captures the covariance between all possible pairs of measurements. The covariance values reflect the noise and redundancy in our measurements.

- In the diagonal terms, by assumption, large values correspond to interesting structure.
- In the off-diagonal terms large magnitudes correspond to high redundancy.

Pretend we have the option of manipulating $\mathbf{C_X}$. We will suggestively define our manipulated covariance matrix $\mathbf{C_Y}$. What features do we want to optimize in $\mathbf{C_Y}$?

D. Diagonalize the Covariance Matrix

We can summarize the last two sections by stating that our goals are (1) to minimize redundancy, measured by the magnitude of the covariance, and (2) maximize the signal, measured by the variance. What would the optimized covariance matrix $\mathbf{C_Y}$ look like?

- All off-diagonal terms in $\mathbf{C_Y}$ should be zero. Thus, $\mathbf{C_Y}$ must be a diagonal matrix. Or, said another way, $\mathbf{Y}$ is decorrelated.
- Each successive dimension in $\mathbf{Y}$ should be rank-ordered according to variance.

There are many methods for diagonalizing $\mathbf{C_Y}$. It is curious to note that PCA arguably selects the easiest method: PCA assumes that all basis vectors $\{\mathbf{p_1}, \ldots, \mathbf{p_m}\}$ are orthonormal, i.e. $\mathbf{P}$ is an *orthonormal matrix*. Why is this assumption easiest?

Envision how PCA works. In our simple example in Figure 2, $\mathbf{P}$ acts as a generalized rotation to align a basis with the axis of maximal variance. In multiple dimensions this could be performed by a simple algorithm:

1. Select a normalized direction in m-dimensional space along which the variance in $\mathbf{X}$ is maximized. Save this vector as $\mathbf{p_1}$.
2. Find another direction along which variance is maximized, however, because of the orthonormality condition, restrict the search to all directions orthogonal to all previous selected directions. Save this vector as $\mathbf{p_i}$
3. Repeat this procedure until m vectors are selected.

The resulting ordered set of $\mathbf{p}$'s are the *principal components*.

In principle this simple algorithm works, however that would bely the true reason why the orthonormality assumption is judicious. The true benefit to this assumption is that there exists

[2] Note that in practice, the covariance σ_{AB}^2 is calculated as $\frac{1}{n-1}\sum_i a_i b_i$. The slight change in normalization constant arises from estimation theory, but that is beyond the scope of this tutorial.

an efficient, analytical solution to this problem. We will discuss two solutions in the following sections.

Notice what we gained with the stipulation of rank-ordered variance. We have a method for judging the importance of the principal direction. Namely, the variances associated with each direction $\mathbf{p_i}$ quantify how "principal" each direction is by rank-ordering each basis vector $\mathbf{p_i}$ according to the corresponding variances.We will now pause to review the implications of all the assumptions made to arrive at this mathematical goal.

E. Summary of Assumptions

This section provides a summary of the assumptions behind PCA and hint at when these assumptions might perform poorly.

I. *Linearity*
Linearity frames the problem as a change of basis. Several areas of research have explored how extending these notions to nonlinear regimes (see Discussion).

II. *Large variances have important structure.*
This assumption also encompasses the belief that the data has a high SNR. Hence, principal components with larger associated variances represent interesting structure, while those with lower variances represent noise. Note that this is a strong, and sometimes, incorrect assumption (see Discussion).

III. *The principal components are orthogonal.*
This assumption provides an intuitive simplification that makes PCA soluble with linear algebra decomposition techniques. These techniques are highlighted in the two following sections.

We have discussed all aspects of deriving PCA - what remain are the linear algebra solutions. The first solution is somewhat straightforward while the second solution involves understanding an important algebraic decomposition.

V. SOLVING PCA USING EIGENVECTOR DECOMPOSITION

We derive our first algebraic solution to PCA based on an important property of eigenvector decomposition. Once again, the data set is $\mathbf{X}$, an $m \times n$ matrix, where m is the number of measurement types and n is the number of samples. The goal is summarized as follows.

> Find some orthonormal matrix $\mathbf{P}$ in $\mathbf{Y} = \mathbf{PX}$ such that $\mathbf{C_Y} \equiv \frac{1}{n}\mathbf{YY}^T$ is a diagonal matrix. The rows of $\mathbf{P}$ are the *principal components* of $\mathbf{X}$.

We begin by rewriting $\mathbf{C_Y}$ in terms of the unknown variable.

$$\begin{aligned}
\mathbf{C_Y} &= \frac{1}{n}\mathbf{YY}^T \\
&= \frac{1}{n}(\mathbf{PX})(\mathbf{PX})^T \\
&= \frac{1}{n}\mathbf{PXX}^T\mathbf{P}^T \\
&= \mathbf{P}(\frac{1}{n}\mathbf{XX}^T)\mathbf{P}^T \\
\mathbf{C_Y} &= \mathbf{PC_X P}^T
\end{aligned}$$

Note that we have identified the covariance matrix of $\mathbf{X}$ in the last line.

Our plan is to recognize that any symmetric matrix $\mathbf{A}$ is diagonalized by an orthogonal matrix of its eigenvectors (by Theorems 3 and 4 from Appendix A). For a symmetric matrix $\mathbf{A}$ Theorem 4 provides $\mathbf{A} = \mathbf{EDE}^T$, where $\mathbf{D}$ is a diagonal matrix and $\mathbf{E}$ is a matrix of eigenvectors of $\mathbf{A}$ arranged as columns.[3]

Now comes the trick. *We select the matrix* $\mathbf{P}$ *to be a matrix where each row* $\mathbf{p_i}$ *is an eigenvector of* $\frac{1}{n}\mathbf{XX}^T$. By this selection, $\mathbf{P} \equiv \mathbf{E^T}$. With this relation and Theorem 1 of Appendix A ($\mathbf{P}^{-1} = \mathbf{P}^T$) we can finish evaluating $\mathbf{C_Y}$.

$$\begin{aligned}
\mathbf{C_Y} &= \mathbf{PC_X P}^T \\
&= \mathbf{P}(\mathbf{E}^T\mathbf{DE})\mathbf{P}^T \\
&= \mathbf{P}(\mathbf{P}^T\mathbf{DP})\mathbf{P}^T \\
&= (\mathbf{PP}^T)\mathbf{D}(\mathbf{PP}^T) \\
&= (\mathbf{PP}^{-1})\mathbf{D}(\mathbf{PP}^{-1}) \\
\mathbf{C_Y} &= \mathbf{D}
\end{aligned}$$

It is evident that the choice of $\mathbf{P}$ diagonalizes $\mathbf{C_Y}$. This was the goal for PCA. We can summarize the results of PCA in the matrices $\mathbf{P}$ and $\mathbf{C_Y}$.

- The principal components of $\mathbf{X}$ are the eigenvectors of $\mathbf{C_X} = \frac{1}{n}\mathbf{XX}^T$.
- The i^{th} diagonal value of $\mathbf{C_Y}$ is the variance of $\mathbf{X}$ along $\mathbf{p_i}$.

In practice computing PCA of a data set $\mathbf{X}$ entails (1) subtracting off the mean of each measurement type and (2) computing the eigenvectors of $\mathbf{C_X}$. This solution is demonstrated in Matlab code included in Appendix B.

---

[3] The matrix $\mathbf{A}$ might have $r \leq m$ orthonormal eigenvectors where r is the rank of the matrix. When the rank of $\mathbf{A}$ is less than m, $\mathbf{A}$ is *degenerate* or all data occupy a subspace of dimension $r \leq m$. Maintaining the constraint of orthogonality, we can remedy this situation by selecting $(m-r)$ additional orthonormal vectors to "fill up" the matrix $\mathbf{E}$. These additional vectors do not effect the final solution because the variances associated with these directions are zero.

VI. A MORE GENERAL SOLUTION USING SVD

This section is the most mathematically involved and can be skipped without much loss of continuity. It is presented solely for completeness. We derive another algebraic solution for PCA and in the process, find that PCA is closely related to singular value decomposition (SVD). In fact, the two are so intimately related that the names are often used interchangeably. What we will see though is that SVD is a more general method of understanding *change of basis*.

We begin by quickly deriving the decomposition. In the following section we interpret the decomposition and in the last section we relate these results to *PCA*.

A. Singular Value Decomposition

Let $\mathbf{X}$ be an arbitrary $n \times m$ matrix[4] and $\mathbf{X}^T\mathbf{X}$ be a rank r, square, symmetric $m \times m$ matrix. In a seemingly unmotivated fashion, let us define all of the quantities of interest.

- $\{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_r\}$ is the set of *orthonormal* $m \times 1$ eigenvectors with associated eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$ for the symmetric matrix $\mathbf{X}^T\mathbf{X}$.

$$(\mathbf{X}^T\mathbf{X})\hat{\mathbf{v}}_i = \lambda_i \hat{\mathbf{v}}_i$$

- $\sigma_i \equiv \sqrt{\lambda_i}$ are positive real and termed the *singular values*.

- $\{\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_r\}$ is the set of $n \times 1$ vectors defined by $\hat{\mathbf{u}}_\mathbf{i} \equiv \frac{1}{\sigma_\mathbf{i}}\mathbf{X}\hat{\mathbf{v}}_\mathbf{i}$.

The final definition includes two new and unexpected properties.

- $\hat{\mathbf{u}}_\mathbf{i} \cdot \hat{\mathbf{u}}_\mathbf{j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$

- $\|\mathbf{X}\hat{\mathbf{v}}_\mathbf{i}\| = \sigma_i$

These properties are both proven in Theorem 5. We now have all of the pieces to construct the decomposition. The scalar version of singular value decomposition is just a restatement of the third definition.

$$\mathbf{X}\hat{\mathbf{v}}_i = \sigma_i \hat{\mathbf{u}}_i \tag{3}$$

This result says a quite a bit. $\mathbf{X}$ multiplied by an eigenvector of $\mathbf{X}^T\mathbf{X}$ is equal to a scalar times another vector. The set of eigenvectors $\{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_r\}$ and the set of vectors $\{\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_r\}$ are both orthonormal sets or bases in r-dimensional space.

We can summarize this result for all vectors in one matrix multiplication by following the prescribed construction in Figure 4. We start by constructing a new diagonal matrix Σ.

$$\Sigma \equiv \begin{bmatrix} \sigma_{\tilde{1}} & & & & \\ & \ddots & & \mathbf{0} & \\ & & \sigma_{\tilde{r}} & & \\ & & & 0 & \\ & \mathbf{0} & & \ddots & \\ & & & & 0 \end{bmatrix}$$

where $\sigma_{\tilde{1}} \geq \sigma_{\tilde{2}} \geq \ldots \geq \sigma_{\tilde{r}}$ are the rank-ordered set of singular values. Likewise we construct accompanying orthogonal matrices,

$$\begin{aligned} \mathbf{V} &= \begin{bmatrix} \hat{\mathbf{v}}_{\tilde{\mathbf{1}}} & \hat{\mathbf{v}}_{\tilde{\mathbf{2}}} & \ldots & \hat{\mathbf{v}}_{\tilde{\mathbf{m}}} \end{bmatrix} \\ \mathbf{U} &= \begin{bmatrix} \hat{\mathbf{u}}_{\tilde{\mathbf{1}}} & \hat{\mathbf{u}}_{\tilde{\mathbf{2}}} & \ldots & \hat{\mathbf{u}}_{\tilde{\mathbf{n}}} \end{bmatrix} \end{aligned}$$

where we have appended an additional $(m-r)$ and $(n-r)$ orthonormal vectors to "fill up" the matrices for $\mathbf{V}$ and $\mathbf{U}$ respectively (i.e. to deal with degeneracy issues). Figure 4 provides a graphical representation of how all of the pieces fit together to form the matrix version of *SVD*.

$$\mathbf{XV} = \mathbf{U}\Sigma$$

where each column of $\mathbf{V}$ and $\mathbf{U}$ perform the scalar version of the decomposition (Equation 3). Because $\mathbf{V}$ is orthogonal, we can multiply both sides by $\mathbf{V}^{-1} = \mathbf{V}^T$ to arrive at the final form of the decomposition.

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T \tag{4}$$

Although derived without motivation, this decomposition is quite powerful. Equation 4 states that *any* arbitrary matrix $\mathbf{X}$ can be converted to an orthogonal matrix, a diagonal matrix and another orthogonal matrix (or a rotation, a stretch and a second rotation). Making sense of Equation 4 is the subject of the next section.

B. Interpreting SVD

The final form of SVD is a concise but thick statement. Instead let us reinterpret Equation 3 as

$$\mathbf{Xa} = k\mathbf{b}$$

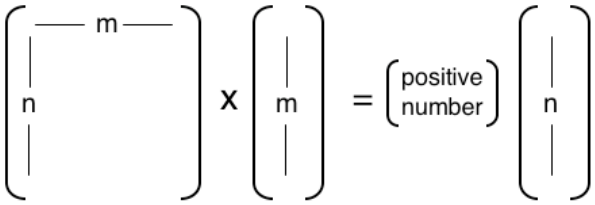
where $\mathbf{a}$ and $\mathbf{b}$ are column vectors and k is a scalar constant. The set $\{\hat{\mathbf{v}}_\mathbf{1}, \hat{\mathbf{v}}_\mathbf{2}, \ldots, \hat{\mathbf{v}}_\mathbf{m}\}$ is analogous to $\mathbf{a}$ and the set $\{\hat{\mathbf{u}}_\mathbf{1}, \hat{\mathbf{u}}_\mathbf{2}, \ldots, \hat{\mathbf{u}}_\mathbf{n}\}$ is analogous to $\mathbf{b}$. What is unique though is that $\{\hat{\mathbf{v}}_\mathbf{1}, \hat{\mathbf{v}}_\mathbf{2}, \ldots, \hat{\mathbf{v}}_\mathbf{m}\}$ and $\{\hat{\mathbf{u}}_\mathbf{1}, \hat{\mathbf{u}}_\mathbf{2}, \ldots, \hat{\mathbf{u}}_\mathbf{n}\}$ are orthonormal sets of vectors which *span* an m or n dimensional space, respectively. In particular, loosely speaking these sets appear to span

[4] Notice that in this section only we are reversing convention from $m \times n$ to $n \times m$. The reason for this derivation will become clear in section 6.3.

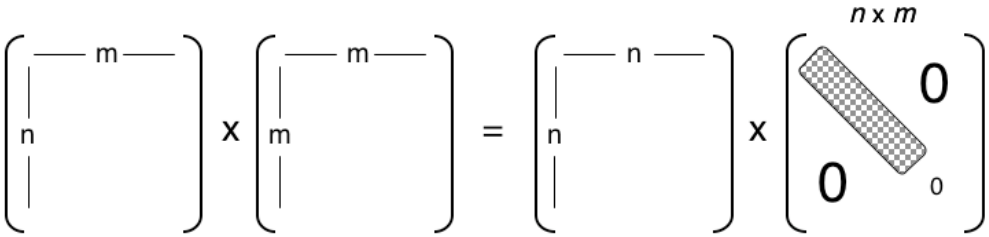
The scalar form of SVD is expressed in equation 3.

$$\mathbf{X}\hat{\mathbf{v}}_i = \sigma_i \hat{\mathbf{u}}_i$$

The mathematical intuition behind the construction of the matrix form is that we want to express all n scalar equations in just one equation. It is easiest to understand this process graphically. Drawing the matrices of equation 3 looks likes the following.



We can construct three new matrices $\mathbf{V}$, $\mathbf{U}$ and Σ. All singular values are first rank-ordered $\sigma_{\tilde{1}} \geq \sigma_{\tilde{2}} \geq \ldots \geq \sigma_{\tilde{r}}$, and the corresponding vectors are indexed in the same rank order. Each pair of associated vectors $\hat{\mathbf{v}}_\mathbf{i}$ and $\hat{\mathbf{u}}_\mathbf{i}$ is stacked in the i^{th} column along their respective matrices. The corresponding singular value σ_i is placed along the diagonal (the ii^{th} position) of Σ. This generates the equation $\mathbf{XV} = \mathbf{U}\Sigma$, which looks like the following.



The matrices $\mathbf{V}$ and $\mathbf{U}$ are $m \times m$ and $n \times n$ matrices respectively and Σ is a diagonal matrix with a few non-zero values (represented by the checkerboard) along its diagonal. Solving this single matrix equation solves all n "value" form equations.

FIG. 4 Construction of the matrix form of SVD (Equation 4) from the scalar form (Equation 3).

all possible "inputs" (i.e. $\mathbf{a}$) and "outputs" (i.e. $\mathbf{b}$). Can we formalize the view that $\{\hat{\mathbf{v}}_\mathbf{1}, \hat{\mathbf{v}}_\mathbf{2}, \ldots, \hat{\mathbf{v}}_\mathbf{n}\}$ and $\{\hat{\mathbf{u}}_\mathbf{1}, \hat{\mathbf{u}}_\mathbf{2}, \ldots, \hat{\mathbf{u}}_\mathbf{n}\}$ span all possible "inputs" and "outputs"?

We can manipulate Equation 4 to make this fuzzy hypothesis more precise.

$$\begin{aligned} \mathbf{X} &= \mathbf{U}\Sigma\mathbf{V}^T \\ \mathbf{U}^T\mathbf{X} &= \Sigma\mathbf{V}^T \\ \mathbf{U}^T\mathbf{X} &= \mathbf{Z} \end{aligned}$$

where we have defined $\mathbf{Z} \equiv \Sigma\mathbf{V}^T$. Note that the previous columns $\{\hat{\mathbf{u}}_\mathbf{1}, \hat{\mathbf{u}}_\mathbf{2}, \ldots, \hat{\mathbf{u}}_\mathbf{n}\}$ are now rows in $\mathbf{U}^T$. Comparing this equation to Equation 1, $\{\hat{\mathbf{u}}_\mathbf{1}, \hat{\mathbf{u}}_\mathbf{2}, \ldots, \hat{\mathbf{u}}_\mathbf{n}\}$ perform the same role as $\{\hat{\mathbf{p}}_\mathbf{1}, \hat{\mathbf{p}}_\mathbf{2}, \ldots, \hat{\mathbf{p}}_\mathbf{m}\}$. Hence, $\mathbf{U}^T$ is a *change of basis* from $\mathbf{X}$ to $\mathbf{Z}$. Just as before, we were transforming column vectors, we can again infer that we are transforming column vectors. The fact that the orthonormal basis $\mathbf{U}^T$ (or $\mathbf{P}$) transforms column vectors means that $\mathbf{U}^T$ is a basis that spans the columns of $\mathbf{X}$. Bases that span the columns are termed the *column space* of $\mathbf{X}$. The column space formalizes the notion of what are the possible "outputs" of any matrix.

There is a funny symmetry to SVD such that we can define a similar quantity - the *row space*.

$$\begin{aligned} \mathbf{XV} &= \Sigma\mathbf{U} \\ (\mathbf{XV})^T &= (\Sigma\mathbf{U})^T \\ \mathbf{V}^T\mathbf{X}^T &= \mathbf{U}^T\Sigma \\ \mathbf{V}^T\mathbf{X}^T &= \mathbf{Z} \end{aligned}$$

where we have defined $\mathbf{Z} \equiv \mathbf{U}^\mathbf{T}\Sigma$. Again the rows of $\mathbf{V}^T$ (or the columns of $\mathbf{V}$) are an orthonormal basis for transforming $\mathbf{X}^T$ into $\mathbf{Z}$. Because of the transpose on $\mathbf{X}$, it follows that $\mathbf{V}$ is an orthonormal basis spanning the *row space* of $\mathbf{X}$. The row space likewise formalizes the notion of what are possible "inputs" into an arbitrary matrix.

We are only scratching the surface for understanding the full implications of SVD. For the purposes of this tutorial though, we have enough information to understand how PCA will fall within this framework.

C. SVD and PCA

It is evident that PCA and SVD are intimately related. Let us return to the original $m \times n$ data matrix $\mathbf{X}$. We can define a

Quick Summary of PCA

1. Organize data as an $m \times n$ matrix, where m is the number of measurement types and n is the number of samples.
2. Subtract off the mean for each measurement type.
3. Calculate the SVD or the eigenvectors of the covariance.

FIG. 5 A step-by-step instruction list on how to perform principal component analysis

new matrix $\mathbf{Y}$ as an $n \times m$ matrix.[5]

$$\mathbf{Y} \equiv \frac{1}{\sqrt{n}}\mathbf{X}^T$$

where each *column* of $\mathbf{Y}$ has zero mean. The choice of $\mathbf{Y}$ becomes clear by analyzing $\mathbf{Y}^T\mathbf{Y}$.

$$\begin{aligned}\mathbf{Y}^T\mathbf{Y} &= \left(\frac{1}{\sqrt{n}}\mathbf{X}^T\right)^T\left(\frac{1}{\sqrt{n}}\mathbf{X}^T\right)\\ &= \frac{1}{n}\mathbf{X}\mathbf{X}^T\\ \mathbf{Y}^T\mathbf{Y} &= \mathbf{C}_\mathbf{X}\end{aligned}$$

By construction $\mathbf{Y}^T\mathbf{Y}$ equals the covariance matrix of $\mathbf{X}$. From section 5 we know that the principal components of $\mathbf{X}$ are the eigenvectors of $\mathbf{C}_\mathbf{X}$. If we calculate the SVD of $\mathbf{Y}$, the columns of matrix $\mathbf{V}$ contain the eigenvectors of $\mathbf{Y}^T\mathbf{Y} = \mathbf{C}_\mathbf{X}$. *Therefore, the columns of* $\mathbf{V}$ *are the principal components of* $\mathbf{X}$. This second algorithm is encapsulated in Matlab code included in Appendix B.

What does this mean? $\mathbf{V}$ spans the row space of $\mathbf{Y} \equiv \frac{1}{\sqrt{n}}\mathbf{X}^T$. Therefore, $\mathbf{V}$ must also span the column space of $\frac{1}{\sqrt{n}}\mathbf{X}$. We can conclude that finding the principal components amounts to finding an orthonormal basis that spans the *column space* of $\mathbf{X}$.[6]

VII. DISCUSSION

Principal component analysis (PCA) has widespread applications because it reveals simple underlying structures in complex data sets using analytical solutions from linear algebra. Figure 5 provides a brief summary for implementing PCA.

A primary benefit of PCA arises from quantifying the importance of each dimension for describing the variability of a data set. In particular, the measurement of the variance along each principle component provides a means for comparing the relative importance of each dimension. An implicit hope behind employing this method is that the variance along a small number of principal components (i.e. less than the number of measurement types) provides a reasonable characterization of the complete data set. This statement is the precise intuition behind any method of *dimensional reduction* – a vast arena of active research. In the example of the spring, PCA identifies that a majority of variation exists along a single dimension (the direction of motion $\hat{\mathbf{x}}$), eventhough 6 dimensions are recorded.

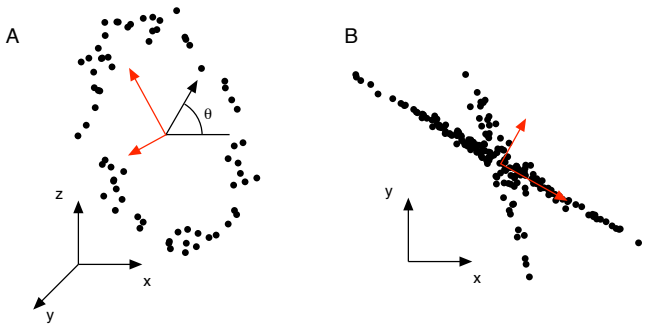


FIG. 6 Example of when PCA fails (red lines). (a) Tracking a person on a ferris wheel (black dots). All dynamics can be described by the phase of the wheel θ, a non-linear combination of the naive basis. (b) In this example data set, non-Gaussian distributed data and non-orthogonal axes causes PCA to fail. The axes with the largest variance do not correspond to the appropriate answer.

Although PCA "works" on a multitude of real world problems, any diligent scientist or engineer must ask *when does PCA fail?* Before we answer this question, let us note a remarkable feature of this algorithm. PCA is completely *non-parametric*: any data set can be plugged in and an answer comes out, requiring no parameters to tweak and no regard for how the data was recorded. From one perspective, the fact that PCA is non-parametric (or plug-and-play) can be considered a positive feature because the answer is unique and independent of the user. From another perspective the fact that PCA is agnostic to the source of the data is also a weakness. For instance, consider tracking a person on a ferris wheel in Figure 6a. The data points can be cleanly described by a single variable, the precession angle of the wheel θ, however PCA would fail to recover this variable.

A. Limits and Statistics of Dimensional Reduction

A deeper appreciation of the limits of PCA requires some consideration about the underlying assumptions and in tandem, a more rigorous description of the source of data. Generally speaking, the primary motivation behind this method is to decorrelate the data set, i.e. remove second-order dependencies. The manner of approaching this goal is loosely akin to how one might explore a town in the Western United States: drive down the longest road running through the town. When

[5] $\mathbf{Y}$ is of the appropriate $n \times m$ dimensions laid out in the derivation of section 6.1. This is the reason for the "flipping" of dimensions in 6.1 and Figure 4.

[6] If the final goal is to find an orthonormal basis for the coulmn space of $\mathbf{X}$ then we can calculate it directly without constructing $\mathbf{Y}$. By symmetry the columns of $\mathbf{U}$ produced by the SVD of $\frac{1}{\sqrt{n}}\mathbf{X}$ must also be the principal components.

one sees another big road, turn left or right and drive down this road, and so forth. In this analogy, PCA requires that each new road explored must be perpendicular to the previous, but clearly this requirement is overly stringent and the data (or town) might be arranged along non-orthogonal axes, such as Figure 6b. Figure 6 provides two examples of this type of data where PCA provides unsatisfying results.

To address these problems, we must define what we consider optimal results. In the context of dimensional reduction, one measure of success is the degree to which a reduced representation can predict the original data. In statistical terms, we must define an error function (or loss function). It can be proved that under a common loss function, mean squared error (i.e. L_2 norm), PCA provides the optimal reduced representation of the data. This means that selecting orthogonal directions for principal components is the best solution to predicting the original data. Given the examples of Figure 6, how could this statement be true? Our intuitions from Figure 6 suggest that this result is somehow misleading.

The solution to this paradox lies in the goal we selected for the analysis. The goal of the analysis is to decorrelate the data, or said in other terms, the goal is to remove second-order dependencies in the data. In the data sets of Figure 6, higher order dependencies exist between the variables. Therefore, removing second-order dependencies is insufficient at revealing all structure in the data.[7]

Multiple solutions exist for removing higher-order dependencies. For instance, if prior knowledge is known about the problem, then a nonlinearity (i.e. *kernel*) might be applied to the data to transform the data to a more appropriate naive basis. For instance, in Figure 6a, one might examine the polar coordinate representation of the data. This parametric approach is often termed *kernel PCA*.

Another direction is to impose more general statistical definitions of dependency within a data set, e.g. requiring that data along reduced dimensions be *statistically independent*. This class of algorithms, termed, *independent component analysis* (ICA), has been demonstrated to succeed in many domains where PCA fails. ICA has been applied to many areas of signal and image processing, but suffers from the fact that solutions are (sometimes) difficult to compute.

Writing this paper has been an extremely instructional experience for me. I hope that this paper helps to demystify the motivation and results of PCA, and the underlying assumptions behind this important analysis technique. Please send me a note if this has been useful to you as it inspires me to keep writing!

[7] When are second order dependencies sufficient for revealing all dependencies in a data set? This statistical condition is met when the first and second order statistics are *sufficient statistics* of the data. This occurs, for instance, when a data set is Gaussian distributed.

Appendix A: Linear Algebra

This section proves a few unapparent theorems in linear algebra, which are crucial to this paper.

1. The inverse of an orthogonal matrix is its transpose.

Let $\mathbf{A}$ be an $m \times n$ orthogonal matrix where $\mathbf{a_i}$ is the i^{th} column vector. The ij^{th} element of $\mathbf{A}^T\mathbf{A}$ is

$$(\mathbf{A}^T\mathbf{A})_{ij} = \mathbf{a_i}^T\mathbf{a_j} = \begin{cases} 1 & if\ i = j \\ 0 & otherwise \end{cases}$$

Therefore, because $\mathbf{A}^T\mathbf{A} = \mathbf{I}$, it follows that $\mathbf{A}^{-1} = \mathbf{A}^T$.

2. For any matrix A, $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ are symmetric.

$$\begin{aligned} (\mathbf{A}\mathbf{A}^T)^T &= \mathbf{A}^{TT}\mathbf{A}^T = \mathbf{A}\mathbf{A}^T \\ (\mathbf{A}^T\mathbf{A})^T &= \mathbf{A}^T\mathbf{A}^{TT} = \mathbf{A}^T\mathbf{A} \end{aligned}$$

3. A matrix is symmetric if and only if it is orthogonally diagonalizable.

Because this statement is bi-directional, it requires a two-part "if-and-only-if" proof. One needs to prove the forward and the backwards "if-then" cases.

Let us start with the forward case. If $\mathbf{A}$ is orthogonally diagonalizable, then $\mathbf{A}$ is a symmetric matrix. By hypothesis, orthogonally diagonalizable means that there exists some $\mathbf{E}$ such that $\mathbf{A} = \mathbf{E}\mathbf{D}\mathbf{E}^T$, where $\mathbf{D}$ is a diagonal matrix and $\mathbf{E}$ is some special matrix which diagonalizes $\mathbf{A}$. Let us compute $\mathbf{A}^T$.

$$\mathbf{A}^T = (\mathbf{E}\mathbf{D}\mathbf{E}^T)^T = \mathbf{E}^{TT}\mathbf{D}^T\mathbf{E}^T = \mathbf{E}\mathbf{D}\mathbf{E}^T = \mathbf{A}$$

Evidently, if $\mathbf{A}$ is orthogonally diagonalizable, it must also be symmetric.

The reverse case is more involved and less clean so it will be left to the reader. In lieu of this, hopefully the "forward" case is suggestive if not somewhat convincing.

4. A symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors.

Let $\mathbf{A}$ be a square $n \times n$ symmetric matrix with associated eigenvectors $\{\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_n}\}$. Let $\mathbf{E} = [\mathbf{e_1}\ \mathbf{e_2}\ \ldots\ \mathbf{e_n}]$ where the i^{th} column of $\mathbf{E}$ is the eigenvector $\mathbf{e_i}$. This theorem asserts that there exists a diagonal matrix $\mathbf{D}$ such that $\mathbf{A} = \mathbf{E}\mathbf{D}\mathbf{E}^T$.

This proof is in two parts. In the first part, we see that the any matrix can be orthogonally diagonalized if and only if it that matrix's eigenvectors are all linearly independent. In the second part of the proof, we see that a symmetric matrix

has the special property that all of its eigenvectors are not just linearly independent but also orthogonal, thus completing our proof.

In the first part of the proof, let $\mathbf{A}$ be just some matrix, not necessarily symmetric, and let it have independent eigenvectors (i.e. no degeneracy). Furthermore, let $\mathbf{E} = [\mathbf{e_1}\ \mathbf{e_2}\ \ldots\ \mathbf{e_n}]$ be the matrix of eigenvectors placed in the columns. Let $\mathbf{D}$ be a diagonal matrix where the i^{th} eigenvalue is placed in the ii^{th} position.

We will now show that $\mathbf{AE} = \mathbf{ED}$. We can examine the columns of the right-hand and left-hand sides of the equation.

$$\begin{aligned} \text{Left hand side}: \quad \mathbf{AE} &= [\mathbf{Ae_1}\ \mathbf{Ae_2}\ \ldots\ \mathbf{Ae_n}] \\ \text{Right hand side}: \quad \mathbf{ED} &= [\lambda_1\mathbf{e_1}\ \lambda_2\mathbf{e_2}\ \ldots\ \lambda_n\mathbf{e_n}] \end{aligned}$$

Evidently, if $\mathbf{AE} = \mathbf{ED}$ then $\mathbf{Ae_i} = \lambda_i\mathbf{e_i}$ for all i. This equation is the definition of the eigenvalue equation. Therefore, it must be that $\mathbf{AE} = \mathbf{ED}$. A little rearrangement provides $\mathbf{A} = \mathbf{EDE}^{-1}$, completing the first part the proof.

For the second part of the proof, we show that a symmetric matrix always has orthogonal eigenvectors. For some symmetric matrix, let λ_1 and λ_2 be distinct eigenvalues for eigenvectors $\mathbf{e_1}$ and $\mathbf{e_2}$.

$$\begin{aligned} \lambda_1\mathbf{e_1}\cdot\mathbf{e_2} &= (\lambda_1\mathbf{e_1})^T\mathbf{e_2} \\ &= (\mathbf{Ae_1})^T\mathbf{e_2} \\ &= \mathbf{e_1}^T\mathbf{A}^T\mathbf{e_2} \\ &= \mathbf{e_1}^T\mathbf{Ae_2} \\ &= \mathbf{e_1}^T(\lambda_2\mathbf{e_2}) \\ \lambda_1\mathbf{e_1}\cdot\mathbf{e_2} &= \lambda_2\mathbf{e_1}\cdot\mathbf{e_2} \end{aligned}$$

By the last relation we can equate that $(\lambda_1 - \lambda_2)\mathbf{e_1}\cdot\mathbf{e_2} = 0$. Since we have conjectured that the eigenvalues are in fact unique, it must be the case that $\mathbf{e_1}\cdot\mathbf{e_2} = 0$. Therefore, the eigenvectors of a symmetric matrix are orthogonal.

Let us back up now to our original postulate that $\mathbf{A}$ is a symmetric matrix. By the second part of the proof, we know that the eigenvectors of $\mathbf{A}$ are all orthonormal (we choose the eigenvectors to be normalized). This means that $\mathbf{E}$ is an orthogonal matrix so by theorem 1, $\mathbf{E}^T = \mathbf{E}^{-1}$ and we can rewrite the final result.

$$\mathbf{A} = \mathbf{EDE}^T$$

. Thus, a symmetric matrix is diagonalized by a matrix of its eigenvectors.

5. For any arbitrary $m \times n$ matrix $\mathbf{X}$, the symmetric matrix $\mathbf{X}^T\mathbf{X}$ has a set of orthonormal eigenvectors of $\{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_n\}$ and a set of associated eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. The set of vectors $\{\mathbf{X}\hat{\mathbf{v}}_1, \mathbf{X}\hat{\mathbf{v}}_2, \ldots, \mathbf{X}\hat{\mathbf{v}}_n\}$ then form an orthogonal basis, where each vector $\mathbf{X}\hat{\mathbf{v}}_i$ is of length $\sqrt{\lambda_i}$.

All of these properties arise from the dot product of any two vectors from this set.

$$\begin{aligned} (\mathbf{X}\hat{\mathbf{v}}_\mathbf{i})\cdot(\mathbf{X}\hat{\mathbf{v}}_\mathbf{j}) &= (\mathbf{X}\hat{\mathbf{v}}_\mathbf{i})^T(\mathbf{X}\hat{\mathbf{v}}_\mathbf{j}) \\ &= \hat{\mathbf{v}}_\mathbf{i}^T\mathbf{X}^T\mathbf{X}\hat{\mathbf{v}}_\mathbf{j} \\ &= \hat{\mathbf{v}}_\mathbf{i}^T(\lambda_j\hat{\mathbf{v}}_\mathbf{j}) \\ &= \lambda_j\hat{\mathbf{v}}_\mathbf{i}\cdot\hat{\mathbf{v}}_\mathbf{j} \\ (\mathbf{X}\hat{\mathbf{v}}_\mathbf{i})\cdot(\mathbf{X}\hat{\mathbf{v}}_\mathbf{j}) &= \lambda_j\delta_{ij} \end{aligned}$$

The last relation arises because the set of eigenvectors of $\mathbf{X}$ is orthogonal resulting in the Kronecker delta. In more simpler terms the last relation states:

$$(\mathbf{X}\hat{\mathbf{v}}_\mathbf{i})\cdot(\mathbf{X}\hat{\mathbf{v}}_\mathbf{j}) = \begin{cases} \lambda_j & i = j \\ 0 & i \neq j \end{cases}$$

This equation states that any two vectors in the set are orthogonal.

The second property arises from the above equation by realizing that the length squared of each vector is defined as:

$$\|\mathbf{X}\hat{\mathbf{v}}_\mathbf{i}\|^2 = (\mathbf{X}\hat{\mathbf{v}}_\mathbf{i})\cdot(\mathbf{X}\hat{\mathbf{v}}_\mathbf{i}) = \lambda_i$$

Appendix B: Code

This code is written for Matlab 6.5 (Release 13) from Mathworks[8]. The code is not computationally efficient but explanatory (terse comments begin with a %).

This first version follows Section 5 by examining the covariance of the data set.

```
function [signals,PC,V] = pca1(data)
% PCA1: Perform PCA using covariance.
%     data - MxN matrix of input data
%            (M dimensions, N trials)
%  signals - MxN matrix of projected data
%       PC - each column is a PC
%        V - Mx1 matrix of variances

[M,N] = size(data);

% subtract off the mean for each dimension
mn =  mean(data,2);
data = data - repmat(mn,1,N);

% calculate the covariance matrix
covariance = 1 / (N-1) * data * data';

% find the eigenvectors and eigenvalues
```

[8] http://www.mathworks.com

```
[PC, V] = eig(covariance);

% extract diagonal of matrix as vector
V = diag(V);

% sort the variances in decreasing order
[junk, rindices] = sort(-1*V);
V  = V(rindices);
PC = PC(:,rindices);

% project the original data set
signals = PC' * data;
```

This second version follows section 6 computing PCA through SVD.

```
function [signals,PC,V] = pca2(data)
% PCA2: Perform PCA using SVD.
%     data - MxN matrix of input data
%            (M dimensions, N trials)
%  signals - MxN matrix of projected data
%       PC - each column is a PC
%        V - Mx1 matrix of variances

[M,N] = size(data);

% subtract off the mean for each dimension
mn =  mean(data,2);
data = data - repmat(mn,1,N);

% construct the matrix Y
Y = data' / sqrt(N-1);

% SVD does it all
[u,S,PC] = svd(Y);

% calculate the variances
S = diag(S);
V = S .* S;

% project the original data
signals = PC' * data;
```