Computational Algorithms for Face Alignment and Recognition

Kathleen Ann Bellino

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mathematics

Terry L. Herdman, Chair
Jeffrey T. Borggaard
John A. Burns
Eugene M. Cliff
Belinda B. King
Helena S. Wisniewski

April, 27 2002
Falls Church, Virginia

Keywords: real-time face recognition, alignment, rotation, active contours

Computational Algorithms for
Face Alignment and Recognition

Kathleen Ann Bellino

(ABSTRACT)

Real-time face recognition has recently become available for the government and industry due to developments in face recognition algorithms, human head detection algorithms, and faster/low cost computers. Despite these advances, however, there are still some critical issues that affect the performance of real-time face recognition software. This paper addresses the problem of off-centered and out-of-pose faces in pictures, particularly in regard to the eigenface method for face recognition. We first demonstrate how the representation of faces by the eigenface method, and ultimately the performance of the software depend on the location of the eyes in the pictures. The eigenface method for face recognition is described: specifically, the creation of a face basis using the singular value decomposition, the reduction of dimension, and the unique representation of faces in the basis. Two different approaches for aligning the eyes in images are presented. The first considers the rotation of images using the orthogonal Procrustes Problem. The second approach looks at locating features in images using energy-minimizing active contours. We then conclude with a simple and fast algorithm for locating faces in images. Future research is also discussed.

Acknowledgments

I would like to thank all those who helped make this project possible, especially:

Dr. Herdman, my advisor, for your continued support and encouragement,

Dr. Borggaard, for always answering and fixing my numerical questions and problems,

Dr. Burns, for your great research ideas (next time I promise to listen to you sooner!), and for reminding me to take breaks,

Dr. Wisniewski, for the opportunity to do this project, and for showing me that math actually has real-world applications,

Jerri and Katie, for your LaTeX help, and Lisa, for staying in the office late to read through my paper when it no longer made sense to me, and finally

My San Diego friends, Christina and Daniel, for making me laugh during those late night hours when I would return home frustrated from research, needing to talk with someone, but not wanting to wake my East Coast friends.

Contents

List of Figures **vi**

1 Introduction **1**

2 Eigenface Method **2**

Creating the Eigenface Basis 2

Methodology 4

3 Face Alignment and Pose **7**

4 Rotation of Images **7**

A Simple Rotation Example 9

Results of Rotation Algorithm 11

Analysis of Rotation of Images 14

5 Active Contours **17**

Level Set Methods 17

Calculus of Variations 18

Dynamic Programming 19

The Greedy Algorithm 21

Energy Terms 21

Loops versus Arrays 24

Pseudocode for Head-finding Algorithm 26

Results of the Head-Finding Algorithm 27

6 Conclusions **30**

7 Future Research **31**

Bibliography **33**

Vita **35**

List of Figures

1 Eigenface 1 (left), Eigenface 2 (right) 5

2 Eigenface 20 (left), Eigenface 40 (right) 6

3 Reduced Basis: k=40 (left), k=35 (middle), k=30(right) 6

4 Face Alignment and Pose: Centered (left), Off-centered (middle), Eigenface Representation(right) 7

5 A Simple Rotation Example 10

6 Horizontal Shift: Centered (left), Off-centered (middle), Rotated picture (right) 11

7 Horizontal Shift: Eigenface Representation of Rotated Picture 12

8 Vertical Shift: Centered (left), Shifted down (middle), Rotation (right) . 13

9 Vertical Shift: Rotation of B^T 13

10 Head-Tilt: Centered (left), Head tilted (middle), Rotation (right) 14

11 Rotation of Different People: Proposed identity (left), New picture (right) 15

12 Rotation of Different People: Rotation (left), Eigenface Representation of Rotation (right) 15

13 Neighborhood around v_i 22

14 Locations of Convergence: Collar (left), Chin (middle), A Combination (right) 27

15 More Examples of Locations of Convergence 28

16 Initial Placement: Converges to collar (left), 10 rows up, converges to chin (right) 29

17 Off-centered Pictures: 5×5 neighborhood (left), 7×7 neighborhood (right) 29

18 Curvature Parameter: $\beta = 1.8$ (left), $\beta = 2.3$ (middle), $\beta = 2.8$ (right) . . 29

19 Size of Neighborhood: 3×3 neighborhood (left), 5×5 neighborhood (right) 30

1 Introduction

Real-time face recognition has recently become available for the government and industry due to developments in face recognition algorithms, human head detection algorithms, and faster/low cost computers. Real-time face recognition is sometimes referred to as being "continouous" or "on-line." A video camera connected to a PC first detects a face in view, takes a picture, and then immediately runs the face recognition software. If the face recognition system is an identification system, or sometimes called "one-to-many," then the image captured is compared to faces saved in a database for a match. The picture in the database that gives the minimum error, if the error is also within a specified threshold, is considered a match. If the system is a verification system, or "one-to-one", then the image and a proposed identity are compared. Similarly, if the error is within a specified threshold, then the person is confirmed to be the proposed identity. Real-time face recognition needs to be fast as well as accurate. Image files are generally large and require a large amount of memory for storage and time to process. Faster, more powerful computers have reduced some of the speed and storage issues, but it was the development of efficient face recognition algorithms, such as the eigenface method, that have made face recognition possible in "real-time"[12]. The eigenface method (see [7]) creates a new face basis using a large random sample of faces and the singular value decomposition. The new basis elements are orthogonal, so every face is expressed as a linear combination of the basis. Furthermore, the basis can be reduced to represent the faces in lower dimensions. A description of the formulation of the eigenface basis and the unique representation of faces will follow later in more detail.

The wide variety of applications of real-time face recognition are what make it so attractive to the government and industry. Some possible applications include facility access control, user authentication for ATM machines, airport security, remote access to networks, and aiding in the recovery of missing children and fugitives. Real-time face recognition helps to create a safer, more secure world because of the ability to recognize a person through a physical trait, but it is less-invasive than other biometrics. It is cost-effective, requiring only a video camera and a PC. It also eliminates the need for multiple passwords, PINs, and access cards [12].

Despite the advances in real-time face recognition, there are still some critical issues that affect the performance of the software. Two such related issues are that of alignment and pose. Alignment and pose are concerned with the position of the face in a picture, specifically, if it is centered and the head is straight. The performance of the eigenface method for real-time face recognition depends on the position of the face in the picture. Even under controlled conditions, with the person looking straight at the camera, there

is no guarantee that the face will be in the exact same position every time. The head may be slightly tilted, or shifted horizontally or vertically. The person may also be standing slightly closer to or further from the camera which then alters the size of the face in the picture. Such small displacements in position and alignment can affect whether a person is accepted or rejected by the recognition system. Completely lining up the facial features in a new picture with the same features in the original picture installed in the face recognition database greatly improves the accuracy of the eigenface method. The goal then is to rotate or translate a specific feature, such as the eyes, in a new picture into a standard position before running the recognition software. However, finding the exact location of the eyes in a digital image of gray values and setting them into a standard position fast enough for real-time face recognition is not an easy task.

In this paper, we first describe the eigenface method for face recognition: specifically how the images are processed, the creation of the face basis using singular value decomposition, and the reduction of dimension, which is essential for real-time face recognition. It is important to understand how each person is uniquely represented in the eigenface basis and identified by the software. We then discuss alignment and pose, and demonstrate how they affect the performance of the eigenface method. Two approaches to the problem are presented. The first considers the orthogonal Procrustes Problem on the rotation of images. The second approach uses active contours to locate features in an image. Once features are identified, they can then be translated into a standard position. Results of both methods and future work are also discussed.

2 Eigenface Method

Consider the following scenario: you approach the ATM and insert your card. Next you hear the computer say, "Please look at the camera." Your picture is taken and the black and white photo is then divided into a grid, 240×240 for example. The finer the grid, the more detail in the photo. Each pixel of the grid has a gray value on the scale from 0 to 255, or 8 bits per pixel; 0 represents black and 255 represents white. The 240×240 matrix of gray values is a digital representation of your face. Suppose you belong to a large bank with many clients that use the ATM. Storing all of the faces in a database in the matrix form described above requires a great deal of memory. Trying to access those images for identification or verification is neither possible nor practical in real-time. The eigenface method which we will briefly describe creates a new basis that uniquely represents each face in fewer bytes, thus not only solving the storage problem, but also reducing the amount of computation time to identify a person ([10], [12]).

Creating the Eigenface Basis

Suppose that we have a random sample of n faces in matrix form, say $\mathbb{R}^{r\times s}$, to be used to create a new face basis. First, concatenate the rows of each matrix to form vectors of length m, which we will denote by $\phi_1, \phi_2, \ldots, \phi_n$. Note that $m = r \times s$. Each $\phi_i \in \mathbb{R}^m$ represents a face in the random sample. Typically, $n \ll m$. Let

$$A = \left[\begin{array}{cccc} \phi_1 & \phi_2 & \cdots & \phi_n \end{array} \right] \in \mathbb{R}^{m\times n}.$$

The singular value decomposition of A, which we denote $SVD(A)$, is

$$A = U\Sigma V^T$$

with

$$U = \left[\begin{array}{cccc} u_1, & u_2, & \cdots, & u_m \end{array} \right] \in \mathbb{R}^{m\times m}, V = \left[\begin{array}{cccc} v_1, & v_2, & \cdots, & v_n \end{array} \right] \in \mathbb{R}^{n\times n}$$

and

$$\Sigma = diag(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \mathbb{R}^{m\times n} \quad \text{with} \quad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0.$$

U and V are unitary matrices with orthonormal columns. The σ_i's are the singular values of A. A is also written

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

with u_i and v_i, the i^{th} columns of U and V, respectively. From this representation of A, one can see that the first n columns of U, $\{u_1, u_2, \ldots, u_n\}$, form an orthonormal basis which span the range of A, denoted $Ran(A)$. $Ran(A)$ is the set of all linear combinations of the random sample of faces. If the sample size is large and contains sufficient variation, then $Ran(A)$ should include all faces. Thus, $\{u_1, u_2, \ldots, u_n\}$ are the new basis elements for the face basis, called eigenfaces. Every face can be uniquely represented as a linear combination of the eigenfaces. In other words, if $b \in \mathbb{R}^m$ is the image of a face in vector form, then there are constants $c_1, c_2, \ldots, c_n$ such that

$$b = c_1u_1 + c_2u_2 + ... + c_nu_n.$$

The coefficients $c_1, c_2, \ldots, c_n$ are unique to each person and are determined by computing the inner product of b with each of the first n columns of U

$$\begin{aligned} c_1 &= u_1^T b \\ c_2 &= u_2^T b \\ &\vdots \\ c_n &= u_n^T b. \end{aligned}$$

To better understand the singular value decomposition of A, the significance of the σ_i's, and the corresponding u_i's in the new eigenface basis, think of a matrix $A \in \mathbb{R}^{m \times n}$ as a transformation from $\mathbb{R}^n \to \mathbb{R}^m$. A maps orthonormal vectors $v_i, v_j \in \mathbb{R}^n$ to $\sigma_i u_i$ and $\sigma_j u_j$, respectively for all $i, j = 1, 2, \ldots, n$,

$$Av_i = \sigma_i u_i \qquad \text{and} \qquad Av_j = \sigma_j u_j.$$

The u_i's, for $i = 1, \ldots, m$, are also orthonormal, and as we have already noted, form a new basis for $Ran(A)$. The transformation $A : \mathbb{R}^n \to \mathbb{R}^m$ creates a hyper-ellipse in $\mathbb{R}^m$. In general, the magnitude of σ_i measures the extent to which the new basis points in the direction of u_i. Since $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$, then σ_1 is the largest principal semi-axis of the hyper-ellipse and carries the most weight in defining the elements in $Ran(A)$. The small σ_i values indicate that the corresponding u_i's are less influential in the new basis. Eliminating a particular u_i from the basis is equivalent to collapsing that semi-axis of the hyper-ellipse and reducing the dimensions of basis. As a result, the eigenfaces corresponding to the smallest singular values can be omitted from the basis and still give a good approximation to the face. The number of basis elements to keep depends on the sample size and the "independence" of the columns of A. Now we can represent a sample face $b \in \mathbb{R}^m$ with fewer terms, that is

$$b = c_1 u_1 + c_2 u_2 + ... + c_k u_k.$$

for some $k < n$. In a real-time face recognition identification system, the coefficients $c_1, c_2, \ldots, c_k$ are saved in a database; in a verification system, they are saved on a smart card or with a corresponding PIN or password ([2], [4], [5], [7]).

So how are you recognized when you go to the ATM? You insert your card which contains your unique coefficients. The camera then takes your picture. Suppose $\hat{b} \in \mathbb{R}^m$ is the digital image captured by the camera and concatenated into vector form. New coefficients are found by computing the inner product of $\hat{b}$ with each the first k columns of U. The new coefficients are then compared to the ones saved on your ATM card. If the coefficient percent error, calculated by

$$\text{coefficient error} = \frac{\|saved - new\|_2}{\|saved\|_2} * 100$$

is within a specified threshold, then you are authorized to use the ATM.

Methodology

For experimentation, we created an eigenface basis using 40 photos of faces. These pictures are the originals from a real-time face recognition system that uses the eigenface

method. Before being installed in the system, these pictures were preprocessed by setting the eyes in a specific position. This is done by the photographer when the pictures are taken or manually at a later time. The alignment of the photos is a way of normalizing them so that the faces can be accurately and appropriately approximated in the eigenface basis and the correct coefficients can be saved in the face recognition database. There are several problems, however, with using this basis. First, 40 faces is not nearly large enough to create a good basis with sufficient variation. Second, we did not have the ability to take new pictures of these people. Since the alignment issue particularly deals with new pictures and setting them into a standard position, not being able to generate new data with the same faces is clearly a concern. In several instances throughout the paper, off-centered pictures are "created" by modifying the standard pictures, and then used to test alignment algorithms.

In order to be able to generate new pictures for experimentation, another eigenface basis was created using pictures of Virginia Tech mathematics graduate students and faculty. Unfortunately, these pictures were not manually standardized, so the eyes in each picture are not in the exact same position. As a result, this eigenface basis is not very accurate for approximating faces. The alignment of the eyes in pictures is critical for creating a good face basis. The purpose of this second basis is simply for generating new pictures to test alignment algorithms, and then for comparing coefficient errors.

The images in Figures 1 and 2 are a few examples of eigenfaces from the first basis of the 40 standardized faces. Remember, the eigenfaces, $u_1, u_2, \ldots, u_n$ are elements of $\mathbb{R}^m$, so the pictures in Figures 1 and 2 have been concatenated back into matrix form. Notice

Figure 1: Eigenface 1 (left), Eigenface 2 (right)

Figure 2: Eigenface 20 (left), Eigenface 40 (right)

that the first two eigenfaces look a lot like faces, whereas eigenfaces 20 and 40 in Figure 2, contain a lot more noise. Eigenfaces 1 and 2 correspond to the two largest singular values, so they are the most important basis elements. They have much more distinct facial features. Eigenface 40, however, is the least significant in the basis, and from the picture on the right in Figure 2, one can clearly see that it does not contribute much to the definition of the face.

Now that we can see the importance of the ranking of eigenfaces, consider the following examples in Figure 3 of face approximations in the reduced basis, $\{u_1, u_2, \ldots, u_k\}$ for some $k \leq n$. The picture on the left is a copy of the original picture and can be exactly

Figure 3: Reduced Basis: k=40 (left), k=35 (middle), k=30(right)

represented using all 40 eigenfaces. The middle picture is an approximation with $k = 35$, the elimination of the last five eigenfaces. Notice that some detail is lost, but that it is still a good approximation. The picture on the right is the approximation with the first 30 eigenfaces. Observe that more detail is lost, but she is still fairly recognizable.

3 Face Alignment and Pose

The position of the face in a picture is a critical issue with face recognition software, particularly with techniques like the eigenface method, that depend on the gray values of the image. The following example illustrates how the eigenface method handles slightly off-centered images. Consider the photos of the girl in Figure 4. The picture on the left

Figure 4: Face Alignment and Pose: Centered (left), Off-centered (middle), Eigenface Representation(right)

was one of 40 used to create the first eigenface basis, so the eyes are in the standard position. However, for this experiment, 10 pixels were trimmed from each side. Now suppose her picture is taken again for recognition by the software and that the new photo is slightly off-centered. See the middle picture in Figure 4. This is really the same image, but shifted to the left. Once her picture is taken, the software computes her coefficients in the eigenface basis and compares them to her saved coefficients that correspond to the first, centered picture. The projection of the off-centered image in the eigenface basis is shown in Figure 4 on the right and does not look much like the original. The coefficient percent error is 29%. From this example, it becomes clear that the position of the face, especially the eyes, in a picture affects the representation in the eigenface basis.

4 Rotation of Images

The first approach to solving the alignment problem involves the rotation of images and the orthogonal Procrustes Problem. Given $A \in \mathbb{R}^{r\times s}$, an original image of a face in matrix form and $B \in \mathbb{R}^{r\times s}$, a new picture of the same face, which may be off-centered, the goal is to align B with A. The original image is from the installment into the face recognition system, so the eyes are set in the "standard" position. Recall from geometry that two shapes are similar if one of the shapes can be rotated, reflected, and/or dilated to completely match up with the other. This geometrical idea motivates the question, "Can B be rotated into A?" The orthogonal Procrustes Problem states: given a data matrix $A \in \mathbb{R}^{r\times s}$ and a different data matrix $B \in \mathbb{R}^{r\times s}$ obtained from the same set of experiments, find the orthogonal matrix Q which best rotates B into A, or

$$\min \|A - BQ\|_F \qquad \text{such that} \qquad Q^TQ = I_s. \tag{1}$$

The constraint that Q be orthogonal restricts the transformations of B to only rotations and reflections. Letting Q to be *any* matrix also allows the shape of B to be stretched ([3], [5]).

The orthogonal Procrustes Problem was first solved in 1952, but a much cleaner proof has since been formulated. This approach can be found in [5]. Recall, for any matrix $C \in \mathbb{R}^{m\times n}$,

$$\|C\|_F = tr(C^TC))^{1/2}.$$

Note that $tr(C)$ denotes $trace(C)$. Then,

$$\begin{aligned}
\|A - BQ\|_F^2 &= [(tr(A - BQ)^T(A - BQ))^{1/2}]^2 \\
&= tr(A^TA - A^TBQ - Q^TB^TA + (BQ)^T(BQ)) \\
&= tr(A^TA - (Q^TB^TA)^T - Q^TB^TA + (BQ)^T(BQ)) \\
&= tr(A^TA) - tr((Q^TB^TA)^T) - tr(Q^TB^TA) + tr((BQ)^T(BQ)) \\
&= tr(A^TA) - 2tr(Q^TB^TA) + tr(B^TB).
\end{aligned}$$

Here we use the identities

$$tr((Q^TB^TA)^T) = tr(Q^TB^TA) \qquad \text{and}$$
$$tr((BQ)^T(BQ)) = \|BQ\|_F^2 = \|B\|_F^2 = tr(B^TB).$$

Therefore,

$$\begin{aligned}
\min \quad \|A - BQ\|_F^2 &= \min \quad \{tr(A^TA) - 2tr(Q^TB^TA) + tr(B^TB)\} \\
&= \min \quad \{\|A\|_F^2 - 2tr(Q^TB^TA) + \|B\|_F^2\}
\end{aligned}$$

$\|A\|_F^2 \geq 0$ and $\|B\|_F^2 \geq 0$ and are fixed, so the problem reduces to

$$\max \quad tr(Q^T B^T A) \qquad \text{such that} \qquad Q^T Q = I_s.$$

Consider $C = B^T A$. By $SVD(C)$, we can write $C = U\Sigma V^T$, with U and V orthogonal, and Σ, the diagonal matrix of singular values. Then

$$C = U\Sigma V^T \quad \text{implies} \quad U^T C V = \Sigma$$

which gives

$$U^T (B^T A) V = \Sigma.$$

Let $Z = V^T Q^T U$. Then

$$\begin{aligned} tr(Q^T B^T A) &= tr(Q^T U \Sigma V^T) \\ &= tr(V^T Q^T U \Sigma) \\ &= tr(Z\Sigma) = \sum_{i=1}^{p} z_{ii}\sigma_i \\ &\leq \sum_{i=1}^{p} \sigma_i. \end{aligned}$$

Here we use the property that $tr(XY) = tr(YX)$ for all X, Y. In addition, the matrix Z is othogonal which means $|z_{ij}| \leq 1$ for all i, j. Therefore, choosing $Z = I$ maximizes

$$tr(Z\Sigma) = tr(Q^T B^T A)$$

which implies

$$Z = V^T Q^T U = I$$

and finally,

$$Q = UV^T. \tag{2}$$

The algorithm for computing Q is very straight forward and can be found in [5]. Given $A, B \in \mathbb{R}^{r \times s}$,

$$\begin{aligned} &C = B^T A \\ &\text{Compute } \; C = U\Sigma V^T \\ &Q = UV^T \end{aligned} \tag{3}$$

A Simple Rotation Example

In an effort to understand the orthogonal Procrustes Problem, consider the following simple example with rectangles A and B in the Figure 5. Take note of the vertices and

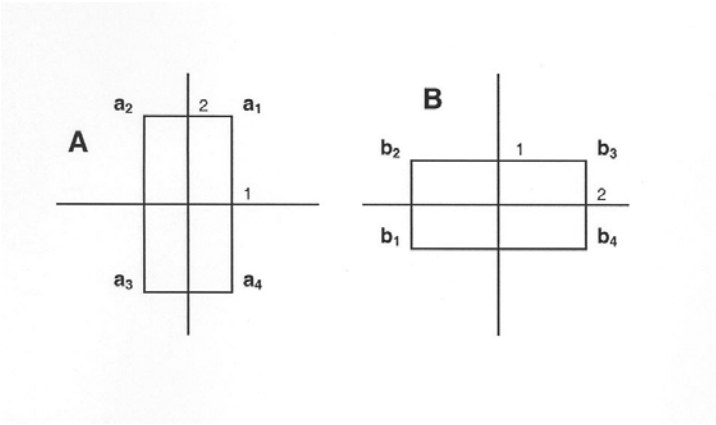

Figure 5: A Simple Rotation Example

the orientation of the rectangles. A and B can be characterized by the vectors that form their vertices.

$$A = \begin{bmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \\ 1 & -2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} -2 & -1 \\ -2 & 1 \\ 2 & 1 \\ 2 & -1 \end{bmatrix}.$$

We seek the orthogonal matrix Q that best rotates B into A. Following the steps in the algorithm (3), we find that

$$Q = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}.$$

But what does Q tell us about the rotation of B into A? Just by looking that the rectangles above, notice that there are two ways to get from rectangle B to rectangle A: either rotate $-90°$ and then reflect about the x-axis or rotate $90°$ and then reflect about the y-axis. Mathematically, the first possibility of rotations and reflections is written as follows

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} = Q.$$

Similarly, for the second set of rotations and reflections, we have

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} = Q.$$

Both sets of reflections and rotations give Q as expected. This simple example demonstrates how the orthogonal Procrustes Problem rotates figures in the plane. We now investigate how it rotates image matrices.

Results of Rotation Algorithm

The rotation algorithm (3) was tested on pictures of students and faculty from the Virginia Tech Mathematics Department as well as some of the 40 pictures from the first eigenface basis. All tests were performed on a PC. Recall, image A is the original picture and B is a new picture. The algorithm finds the optimal orthogonal matrix Q that rotates B into A. Thus, BQ is the new, rotated picture. The results of the rotation of images varied, mostly depending on the position of the face in the pictures. For images that were off-centered, only to the right or left, the algorithm worked extremely well. In other cases, however, details in the face are lost. In order to analyze and understand this rotation of image matrices, we now examine different types of misalignment independently.

The first case to consider is the situation in which the face in the new picture is only shifted to the left or right. For this case, the algorithm performed exceptionally well. Facial features in B were moved to the exact position of features in A. This result was especially pleasing, since the goal is to line up distinctive facial features, specifically the eyes. The following example demonstrates these results using the same pictures in Figure 4. They were originally used to show the projection of an off-centered picture in the eigenface basis. The image on the left is A, the centered, original picture. The

Figure 6: Horizontal Shift: Centered (left), Off-centered (middle), Rotated picture (right)

picture in the middle represents B, the off-centered picture. The third picture on the right is BQ, the rotated picture. Notice that the rotated picture is almost an exact copy of the original. With a little thought, we realize that this result is precisely as we should expect. The optimal Q that rotated B into A is close to a permutation matrix, an identity matrix with the columns (or rows) re-ordered. When B in multiplied by Q on the right side, the columns of B are simply rearranged, thus moving the face in picture B to the exact position of the face in A. Now, the new rotated image is projected into the eigenface basis. Figure 7 shows the image of this projection. With the facial features

Figure 7: Horizontal Shift: Eigenface Representation of Rotated Picture

lined up by the rotation, the eigenface basis does a much better job of approximating the face. The coefficient error is 0.5%, indicating almost a perfect match. This example is ideal and demonstrates the best job that the rotation algorithm can do. In most cases, the face in a new picture is not going to be *exactly* shifted horizontally; it will most likely be shifted slightly up or down, a little closer to or further from the camera, or even tilted. We now look at how the rotation of images handles some of these other cases.

Next, consider the case with B shifted vertically from the original. The rotation algorithm did not do well with faces that were shifted up or down, even with small displacements. Consider the pictures in Figure 8. The same girl is used, only now her face is shifted down. This case is similar to the horizontal shift in that B is simply a rearrangement of A, but it is the rows that are moved, and not the columns. Multiplying B by a permutation matrix on the left side flips the rows around. Thus, we really interested in solving

$$\min \quad \|A - QB\|_F \qquad \text{such that} \qquad Q^T Q = I_r$$

or equivalently,

$$\min \quad \|A^T - B^T Q\|_F \qquad \text{such that} \qquad Q^T Q = I_r.$$

Figure 8: Vertical Shift: Centered (left), Shifted down (middle), Rotation (right)

The algorithm in (3) was run on the pictures in the example above using A^T and B^T in place of A and B, respectively. The results proved to be exactly as expected. The rows of B that represent the face are moved up, and the new, rotated picture is completely lined up with the original picture, (see Figure 9). This vertical shift case presents a

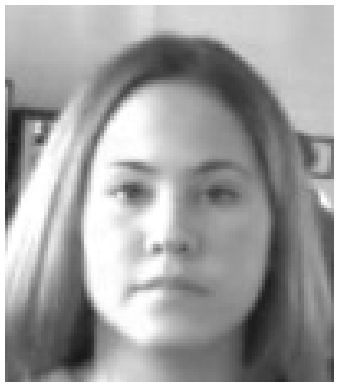

Figure 9: Vertical Shift: Rotation of B^T

problem: when a new picture is taken, the computer does not know whether the picture is shifted horizontally or vertically, and which form of rotation gives the best results.

The cases in which the face was either closer to or further from the camera or the head was tilted did not appear to be promising. Facial features were always brought into the position of features in A, but the faces appeared fuzzy. The accuracy and detail of the rotated image mostly depended on the how different the new picture was from the original. If the head was very tilted, or the face was much bigger or smaller than the original, then the rotated picture usually contained a lot of noise. An example of a small

head tilt and the resulting rotation are shown in Figure 10. Observe the alignment of facial features and loss of details in the face even with such a small displacement from the original. In all cases, however, the coefficient error of the rotated image, BQ with the original, A, was smaller than the coefficient error of B with A.

Figure 10: Head-Tilt: Centered (left), Head tilted (middle), Rotation (right)

In every example up to this point, the critical assumption has been made that the new picture, B, is the *same* person as A, the proposed identity. In other words, we have only tried rotating a person into him/herself. We must now consider the possibility that B is not the same person as A. Can a person be rotated into someone else? If so, then in an effort to solve the alignment problem, we would defeat the whole purpose of face recognition. The initial hope was that the rotation of one person into another would distort the image so much that the new creation would be unrecognizable. When the rotated image is then projected back into the eigenface basis, the coefficient error of the new picture with the proposed identity would be too large to accept. The results thus far have been inconclusive, but not too encouraging. Consider Figures 11 and 12. They demonstrate the rotation of one girl into another. The new rotated picture, on the left in Figure 12, is distorted and appears to be a combination of the two girls. When the rotated picture is approximated in the basis, however, the eigenface representation looks a lot like the first girl! See the picture on the right in Figure 12. The results thus far are deemed inconclusive because this eigenface basis was only created with 40 faces, not nearly large enough for accurate testing.

Figure 11: Rotation of Different People: Proposed identity (left), New picture (right)

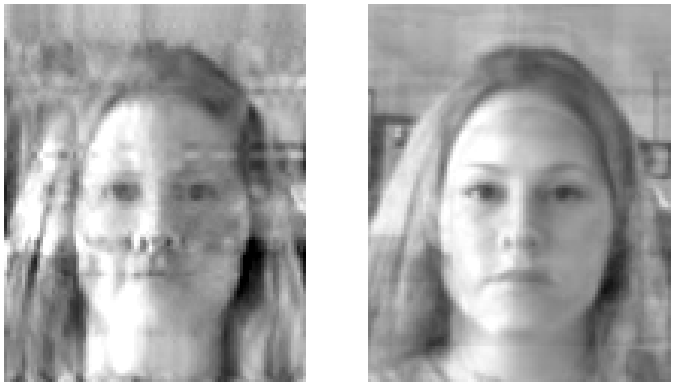

Figure 12: Rotation of Different People: Rotation (left), Eigenface Representation of Rotation (right)

Analysis of Rotation of Images

In all cases, the rotation algorithm finds the optimal orthogonal Q so that BQ is as close to A as possible. The identity matrix is an orthogonal matrix, so

$$\|A - BQ\|_F \leq \|A - BI\|_F = \|A - B\|_F.$$

As a result, the coefficient error between A and BQ is *always* smaller than the coefficient error between A and B. This consequence is not hard to show. First, concatenate the rows of A, B, and BQ. Let $\hat{A}, \hat{B}$, and $\hat{C} \in \mathbb{R}^m$ denote the vector representations of A, B, and BQ, respectively. Assume that $\hat{A}, \hat{B}$, and $\hat{C}$ are elements of the $Ran(U)$. By definition, for $A \in \mathbb{R}^{r \times s}$,

$$\|A\|_F = \sqrt{\sum_{i=1}^{r} \sum_{j=1}^{s} |a_{ij}|^2}$$

which implies that

$$\|A - B\|_F = \|\hat{A} - \hat{B}\|_F \qquad \text{and} \qquad \|A - BQ\|_F = \|\hat{A} - \hat{C}\|_F.$$

Therefore, we have the inequality

$$\|\hat{A} - \hat{C}\|_F \leq \|\hat{A} - \hat{B}\|_F.$$

Now suppose that $a_1, a_2, \ldots, a_n$ are the unique coefficients that correspond to $\hat{A}$, $b_1, b_2, \ldots, b_n$ are the coefficients of $\hat{B}$, and $c_1, c_2, \ldots, c_n$ are the coefficients of $\hat{C}$. Then,

$$\begin{aligned}
\hat{A} &= u_1 a_1 + u_2 a_2 + \ldots + u_n a_n = U\mathbf{a} \\
\hat{B} &= u_1 b_1 + u_2 b_2 + \ldots + u_n b_n = U\mathbf{b} \\
\hat{C} &= u_1 c_1 + u_2 c_2 + \ldots + u_n c_n = U\mathbf{c}
\end{aligned}$$

where $U \in \mathbb{R}^{m \times m}$ with columns $u_1, u_2, \ldots, u_n$, and

$$\begin{aligned}
\mathbf{a} &= [a_1, a_2, \ldots, a_n, 0, \ldots, 0,]^T \in \mathbb{R}^m, \\
\mathbf{b} &= [b_1, b_2, \ldots, b_n, 0, \ldots, 0]^T \in \mathbb{R}^m, \\
\mathbf{c} &= [c_1, c_2, \ldots, c_n, 0, \ldots, 0]^T \in \mathbb{R}^m.
\end{aligned}$$

Recall, $n \ll m$. We then have

$$\begin{aligned}
\|\hat{A} - \hat{B}\|_F = \|U\mathbf{a} - U\mathbf{b}\|_F = \|U(\mathbf{a} - \mathbf{b})\|_F = \|\mathbf{a} - \mathbf{b}\|_F \\
\|\hat{A} - \hat{C}\|_F = \|U\mathbf{a} - U\mathbf{c}\|_F = \|U(\mathbf{a} - \mathbf{c})\|_F = \|\mathbf{a} - \mathbf{c}\|_F
\end{aligned}$$

since U is orthogonal and $\|Ux\|_F = \|x\|_F$ for all $x \in \mathbb{R}^n$. $\|\mathbf{a}-\mathbf{b}\|_F$ and $\|\mathbf{a}-\mathbf{c}\|_F$ are the coefficient errors of A with B and A with BQ, respectively. Therefore, it follows that

$$\|\mathbf{a}-\mathbf{c}\|_F = \|\hat{A}-\hat{C}\|_F \leq \|\hat{A}-\hat{B}\|_F = \|\mathbf{a}-\mathbf{b}\|_F.$$

The rotation of B into A minimizes *all* coefficient errors, whether B is the same person as A or not. The rotation is clearly recognizing similar facial geometry in the two pictures because the common features of eyes, nose, and mouth always line up. The smaller details in the face are lost. Thus, the rotated images, despite loss of fine details, always have a smaller coefficient error. This demonstrates even more the importance of alignment for the eigenface method. It is the little details, however, that distinguish people from one another. If distinctive features, such as the eyes, could be located within an image and then translated into a standard position, we would avoid altering the facial details. But how do we find the eyes in an image matrix of gray values? We now turn to active contours.

5 Active Contours

Active contours are used to find objects and features in gray value images. Two popular techniques are level set methods ([8], [9]) and energy-minimizing snakes ([1], [11]) which are based on calculus of variations. In both methods, an initial contour is placed on the image plane around the object. The contour is then pushed and pulled toward the object based on curvature and the image gradient. Convergence is local and depends on the initial placement of the contour and its proximity to the object. Guessing the approximate position of the eyes in an image matrix of gray values is difficult. If the exact location of the head is known, then approximating the position of the eyes and ultimately finding them becomes an easier task. The new objective becomes locating the head using active contours. In the following paragraph, the first attempt with the level set method is briefly described. Difficulties with the numerics of this method led to the energy-minimizing snake technique.

Level Set Methods

The mathematics of the level set method is to define a function ϕ on the whole plane which depends on a closed moving contour, $\gamma(t)$. Let (x, y) be a point on the plane. Define

$$\phi(t, x, y) = \pm d$$

where d is the distance from the point (x, y) to the contour $\gamma(t)$ at time t. Distance d is negative if (x, y) is inside the contour and positive if (x, y) is outside the contour. The level set $\phi = 0$ at time t corresponds to the position of the moving contour $\gamma(t)$, so it must be that

$$\phi(x(t), t) = 0 \tag{4}$$

where $x(t), t > 0$ is the path of a point on γ. The contour moves in the direction normal to the curve at a speed that depends on the curvature and the image gradient. Points on the contour take big steps when the image gradient is small and the gray values are uniform. Smaller steps are taken when the image gradient is large and the contour is close to an edge. Intuitively, the motion of the contour makes sense, but implementation is more complicated than it first appears. The evolution of ϕ, derived from (4), is given by

$$\phi_t + F|\nabla\phi| = 0 \tag{5}$$

where F is the speed function. Solving the partial differential equation in (5) using the upwind finite difference scheme introduced some potential numerical instabilities. In addition, this method is computationally expensive. First, since the speed function F is only defined on γ, we must calculate its global extension so that all the level sets are moved. Then, in an effort to keep the level sets equally spaced, the set $\phi = 0$ must be recomputed every few iterations by calculating the distance from every point on the plane to the contour $\gamma(t)$ (see [8], [9]). These and other difficulties with this method led to the energy-minimizing active contour method ([8], [9]).

Calculus of Variations

Energy-minimizing active contours, or snakes, based on calculus of variations theory and techniques, are also used to find objects in images. Calculus of variations problems are concerned with minimizing a functional of the form

$$J(y) = \int_a^b F(x, y, y')dx \tag{6}$$

for some objective function $F(x, y, y')$. The goal is to find a curve y_* such that $J(y_*) \leq J(y)$ for all y in a neighborhood of y_*. If such a y_* exists, then it is called a weak local minimizer of J. Necessary and sufficient conditions, such as the Euler Necessary Condition and Jacobi Necessary Condition are used to find extremals of the functional. Kass, Terzopoulos, and Witkin [6] derived an objective function $E(v(s))$ that describes

the total energy of a contour where s is arc length and $v(s) = (x(s), y(s))$ is the parameterization of the contour. The energy functional is

$$E_{total} = \int_0^1 \{E_{int}(v(s)) + E_{image}(v(s)) + E_{con}(v(s))\}\, ds \tag{7}$$

with $E_{int}(v(s)) = \alpha(s)|v_s(s)|^2 + \beta(s)|v_{ss}(s)|^2$, the internal energy of the contour. Notice E_{int} has first and second order continuity terms that help keep the contour smooth. E_{image} is an image energy term that attracts the contour to edges. E_{con} is the constraint energy that "represents the energy of a spring connected between a point on the contour and some point in the plane"(see [1]). For the objective function $E(s, v_s, v_{ss})$, the Euler Necessary Condition is

$$E_s - \frac{\partial}{\partial s} E_{v_s} + \frac{\partial^2}{\partial s^2} E_{v_{ss}} = 0. \tag{8}$$

Letting $E_{ext} = E_{image} + E_{con}$, yields the objective energy function

$$E(s, v_s, v_{ss}) = \alpha(s)|v_s(s)|^2 + \beta(s)|v_{ss}(s)|^2 + E_{ext}(v(s)). \tag{9}$$

Next, we apply the Euler Necessary Condition in (8) to (9) which gives the following equations

$$-\alpha x_{ss} + \beta x_{ssss} + \frac{\partial E_{ext}}{\partial x} = 0$$
$$-\alpha y_{ss} + \beta y_{ssss} + \frac{\partial E_{ext}}{\partial y} = 0.$$

Now, the basic idea is to discretize the above equations and solve for candidate minimizers using iterative methods such as Jacobi or Gauss-Seidel ([1], [6]).

Dynamic Programming

There are several problems with using this variational calculus method, which Amini, Jain, and Weymouth, [1], point out in "Using Dynamic Programming for Solving Variational Problems in Vision." First, numerical instability may occur when approximating such high order derivatives on discrete, and perhaps noisy, image data. Second, there is a possibility that the active contour will converge to a maximum or a stationary point. The necessary conditions do not guarantee convergence to a minimum, only an extremal. Lastly, applying hard constraints to the problem adds more variables, typically making

it more difficult. To resolve some of these issues, Amini, et al, [1], suggest using dynamic programming to minimize the energy functional. Before describing this dynamic programming method, we first observe the connection between dynamic programming and the calculus of variations,

> The approach to dynamical programming is to solve the optimization problem by studying a collection, or family of problems containing a particular problem as a member. This is known as embedding. Dynamic programming regards the extremal as an envelope of tangents, and attempts to determine the optimal direction at each point on the extremal (see [1]).

The dynamic programming algorithm minimizes the energy functional

$$E_{total} = \int E(v(s))ds. \tag{10}$$

It can be discretely approximated by

$$E_{total} = \sum_{i=1}^{n} E(v_i)$$

where $v_1, v_2, \ldots, v_n$ are points on the contour. A "discrete multistage decision process" is then used to find the contour with minimum energy by

$$E(v_1, v_2, \ldots, v_n) = E_1(v_1, v_2) + E_2(v_2, v_3) + \cdots + E_{n-1}(v_{n-1}, v_n)$$

for an energy functional with only the first order continuity term or

$$E(v_1, v_2, \ldots, v_n) = E_1(v_1, v_2, v_3) + E_2(v_2, v_3, v_4) + \cdots + E_{n-2}(v_{n-2}, v_{n-1}, v_n)$$

if the second order term is included. Each point on the contour, $v_1, v_2, \ldots, v_n$ can move around in a neighborhood of m points. The contour converges when the total energy remains the same between two iterations (see [1]).

There are some attractive qualities of this dynamic programming algorithm that the authors point out [1]. First, convergence to a local minimum is guaranteed. Recall, the necessary conditions in the calculus of variations technique only assured an extremal. Second, the contour progresses along discrete points of the grid of gray values. This is a particularly appealing quality because it takes advantage of the structure of image data. A third attractive feature of the dynamic programming method is that it has the ability to easily add hard constraints. In fact, additional constraints often simplify the problem

by limiting the possibilities of the decision process. Recall, in the variational calculus method, hard constraints further complicated the problem. This dynamic programming method of minimizing of the energy functional, however, has one major weakness; it is computationally very expensive. On a contour with n points where each point has m possibilities, and only using the first order continuity term, each iteration is $O(nm^2)$. Adding the second order curvature term, the algorithm increases to $O(nm^3)$. In addition, expanding the size neighborhood around each v_i, significantly lengthens computation time. Despite the attractive qualities of the dynamic programming algorithm, it is computationally too expensive for our purpose of real-time face recognition (see [1],[11]).

The Greedy Algorithm

Williams and Shah, in "A Fast Algorithm for Active Contours and Curvature Estimation," [11], identify the strengths and weaknesses of the dynamic programming and the calculus of variations methods, and then present the greedy algorithm, which adapts strengths of both methods. The greedy algorithm is numerically stable and flexible like the dynamic programming method, but is much faster. It minimizes

$$E = \int \{\alpha(s)E_{cont} + \beta(s)E_{curv} + \gamma(s)E_{image}\}\, ds, \tag{11}$$

a similar functional to (7), but eliminates E_{con}, the constraint energy. E_{cont} and E_{curv} are the continuity and curvature energy terms that correspond to E_{int}, the internal energy. E_{image} is the external energy term which measures the image gradient. The parameters $\alpha(s), \beta(s)$, and $\gamma(s)$ are the weights for the corresponding energy terms.

The greedy algorithm is iterative and time-delayed like the dynamic programming method, but computationally much less expensive. At each point, v_i, on the contour, the greedy algorithm follows this process: the three energy terms E_{cont}, E_{curv}, and E_{image} are calculated for all points in a neighborhood of v_i using v_{i-1} and v_{i+1}. v_i moves around to the positions of its neighbors, while v_{i-1} and v_{i+1} remain fixed. See Figure 13. The three terms are summed for all positions of v_i. The point in the neighborhood that has minimum energy becomes the new position of v_i. Then, move to v_{i+1} and repeat the process. Note that if the contour is closed, then the procedure is followed for $i = 1, \ldots, n-1$ with the first and last two points on the contour overlapping.

Energy Terms

The head-finding algorithm is based on the greedy algorithm but altered for our purposes. The basic procedure is the same as the greedy algorithm: the energy is computed in a neighborhood of v_i, while v_{i-1} and v_{i+1} stay fixed. The location that yields minimum energy becomes the new position of v_i. The same energy functional in (11) is minimized,

$$E(v(s)) = \int \{\alpha E_{cont}(v(s)) + \beta E_{curv}(v(s)) + \gamma E_{image}(v(s))\}\, ds.$$

The goal is to find a contour v_* such that $E(v_*) \leq E(v)$ for all v in a neighborhood of v_*. We hope that the curve v_* with minimum energy corresponds to the outline of the head. Now, we describe the three energy terms in detail and how they are computed.

E_{cont} is the first of the internal energy terms and helps keep the contour smooth. In the dynamic programming algorithm, the term

$$|v_s|^2 = |v_{i-1} - v_i|^2 \tag{12}$$

is used. The authors of the greedy algorithm, [11], recognized that this term was actually forcing the neighboring points to come together by minimizing the distance between the points. They suggest using the term,

$$\bar{d} - |v_i - v_{i-1}| \tag{13}$$

where $\bar{d}$ is the average distance between neighboring points on the contour. This first order continuity term in (13) is minimized when the distance between points is close to the average, thus encouraging points on the contour to stay equally spaced. At the end of each iteration, a new $\bar{d}$ is computed. This continuity term (13), used in the greedy algorithm, is more appropriate for object recognition algorithms that are concerned with discovering and defining the *shape* of an object rather than its *location*. In the examples demonstrating the greedy algorithm in [11], the initial contour was placed close enough

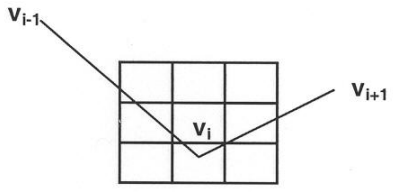

Figure 13: Neighborhood around v_i

to the object that an edge was sure to be detected within a few iterations. Thus, the continuity term in (12) caused the contour to shrink and lose the definition of the object, especially at corners. In our object detection problem, we do not know the exact location of the head in our pictures, so we cannot place the initial contour as carefully. In addition, we are never concerned with the formation of corners. As a result, we are motivated to use the continuity term in (12)

$$E_{cont} = |v_{i-1} - v_i|^2$$

in the head-finding algorithm. The E_{cont} values are calculated for all the points in the neighborhood of v_i, while v_{i-1} on the contour remains fixed. All the values are then divided by the maximum E_{cont} value in the neighborhood, placing them between 0 and 1 to normalize them.

The second energy term E_{curv} controls the curvature of the contour as it evolves. The curvature at v_i is calculated discretely by

$$E_{curve} = |v_{i-1} - 2v_i + v_{i+1}|^2,$$

which is an approximation to the second derivative along the path. Just as the E_{cont} terms were computed and normalized, at each position around v_i, E_{curv} is calculated and then divided by the maximum in the neighborhood.

The last energy term, E_{image}, is the external energy and is what attracts the contour to edges. Let $G(x_i, y_i)$ denote the gray value at $v_i = (x_i, y_i)$. The norm of the image gradient at a point is calculated by

$$I(v_i) = I(x_i, y_i) = \left[\left(\frac{G(x_i + 1, y_i) - G(x_i - 1, y_i)}{2} \right)^2 + \left(\frac{G(x_i, y_i + 1) - G(x_i, y_i - 1)}{2} \right)^2 \right]^{\frac{1}{2}}.$$

The image energy is then defined to be

$$E_{image} = \frac{min - I(v_i)}{max - min}. \tag{14}$$

In this case, min and max are the smallest and largest image gradients in the neighborhood of v_i, respectively. Subtracting $I(v_i)$ from the min and then dividing by the difference of the min and max is a way of normalizing this energy term. Note that E_{image} is negative and that points with high image gradients will have the most negative values. The greedy algorithm [11] includes an extra step to help the contour move along areas of roughly constant image gradients. Consider the example of a point v_i on the contour such that the neighborhood of v_i contains gradient values of 20, 21 and 22. The resulting E_{image} values are $0, -0.5$ and -1.0 and do not reflect the uniformity of the gradient over this region. To resolve this problem, the following step is added:

$$\text{if } max - min < 5, \text{ then } min = max - 5$$

With this additional constraint, the E_{image} values become $-.6, -.8$, and -1.0. This situation is more likely to occur far from edges, thus it helps points that are further from the object to move. This step is included in the head-finding algorithm to deter points from settling at other possible local minimums that may be far from the head.

At the end at each iteration, the greedy algorithm recomputes the curvature at every point to detect the formation of corners. Points reporting high curvature and a high image gradient are determined to be corners and the β_i parameter is adjusted accordingly. This step is excluded from the head-finding algorithm because we are never interested in corners forming. Once the three energy terms, E_{cont}, E_{curve}, and E_{image}, are computed and normalized, the parameters α, β, and γ are then used as weights for the corresponding energy terms.

Loops versus Arrays

To compute the energy in the neighborhood around v_i and to determine if v_i should move to a position of less energy, the greedy algorithm uses the following loop

for $j = 0$ to $m - 1$ (m is the number of points in the neighborhood)
  $E_j = \alpha E_{cont,j} + \beta E_{curv,j} + \gamma E_{image,j}$
  if $E_j < E_{min}$, then
    $E_{min} = E_j$
    $jmin = j$
  end
  Move v_i to location $jmin$
end.

Loops are computationally inefficient in interpreted environments, such as MATLAB, because terms are computed one at a time. In addition, increasing the size of the neighborhood around v_i when using loops, significantly lengthens the amount of time to run the algorithm. Algorithms for real-time face recognition need to be as fast as possible. The size of the neighborhood around v_i is very important to the face-find

problem because the exact location of the head in our pictures is not known. The placement of the initial contour and its proximity to the head cannot be controlled, so the contour needs to be able to travel accross far distances. In an effort to make the head-finding algorithm faster and to easily be able to increase the size of the neighborhood, the energy computations around v_i are reformulated to use arrays instead of loops. The following example demonstrates how the head-finding algorithm computes the energy terms in a 3×3 neighborhood of v_i. A 3×3 neighborhood around $v_i = (x_i, y_i)$ can be represented by the following array with v_i as the center point

$$\begin{bmatrix} (x_i - 1, y_i - 1) & (x_i - 1, y_i) & (x_i - 1, y_i + 1) \\ (x_i, y_i - 1) & (x_i, y_i) & (x_i, y_i + 1) \\ (x_i + 1, y_i - 1) & (x_i + 1, y_i) & (x_i + 1, y_i + 1) \end{bmatrix}.$$

The neighborhood is then separated into x and y components, denoted by X and Y:

$$X = \begin{bmatrix} x_i - 1 & x_i - 1 & x_i - 1 \\ x_i & x_i & x_i \\ x_i + 1 & x_i + 1 & x_i + 1 \end{bmatrix}, \; Y = \begin{bmatrix} y_i - 1 & y_i & y_i + 1 \\ y_i - 1 & y_i & y_i + 1 \\ y_i - 1 & y_i & y_i + 1 \end{bmatrix}.$$

Recall, that $v_{i-1} = (x_{i-1}, y_{i-1})$ stays fixed during computations in a neighborhood of v_i, so let

$$X_{-1} = \begin{bmatrix} x_{i-1} & x_{i-1} & x_{i-1} \\ x_{i-1} & x_{i-1} & x_{i-1} \\ x_{i-1} & x_{i-1} & x_{i-1} \end{bmatrix}, \; Y_{-1} = \begin{bmatrix} y_{i-1} & y_{i-1} & y_{i-1} \\ y_{i-1} & y_{i-1} & y_{i-1} \\ y_{i-1} & y_{i-1} & y_{i-1} \end{bmatrix}.$$

X_{-1} and Y_{-1} are the arrays that correspond to the x and y components of v_{i-1}. Similarly, let

$$X_{+1} = \begin{bmatrix} x_{i+1} & x_{i+1} & x_{i+1} \\ x_{i+1} & x_{i+1} & x_{i+1} \\ x_{i+1} & x_{i+1} & x_{i+1} \end{bmatrix}, \; Y_{+1} = \begin{bmatrix} y_{i+1} & y_{i+1} & y_{i+1} \\ y_{i+1} & y_{i+1} & y_{i+1} \\ y_{i+1} & y_{i+1} & y_{i+1} \end{bmatrix}$$

correspond to v_{i+1}. With this formulation, one can calculate

$$E_{cont} = |v_{i-1} - v_i|^2 = [(x_{i-1} - x_i)^2 + (y_{i-1} - y_i)^2]^{\frac{1}{2}}$$

at each point in the neighborhood at the same time by

$$E_{cont} = [(X_{-1} - X)^2 + (Y_{-1} - Y)^2]^{\frac{1}{2}}.$$

Note that the squares and square root of the matrices are calculated componentwise. It becomes easy to calculate the curvature energy term as well by

$$E_{curv} = (X_{-1} - 2X + X_{+1})^2 + (Y_{-1} - 2Y + Y_{+1})^2.$$

The image gradient at each point in the neighborhood is represented by the following array

$$\begin{bmatrix} I(x_i - 1, y_i - 1) & I(x_i - 1, y_i) & I(x_i - 1, y_i + 1) \\ I(x_i, y_i - 1) & I(x_i, y_i) & I(x_i, y_i + 1) \\ I(x_i + 1, y_i - 1) & I(x_i + 1, y_i) & I(x_i + 1, y_i + 1) \end{bmatrix}.$$

Once the min and max of image gradient are determined, it is not hard to calculate E_{image} by (14). Each of the energy terms is normalized and then all three are summed to form E_{total}, a 3×3 array containing the total energy of v_i and each of its neighbors. The position of the smallest component of E_{total} is the location of the minimum energy and becomes the new location of v_i. The pseudocode below illustrates the basic structure of the head-finding algorithm.

Pseudocode for Head-finding Algorithm

Initialize weights α, β, and γ.
Initialize m ($m \times m$ is the size of the matrix neighborhood around v_i).
Read in the image and send it through the filter to smooth out the edges.
Create image gradient matrix for all points in the image.
Define the initial contour.

while ($pts\_moved > pts\_moved\_threshold$)
  $pts\_moved = 0$
  for $j = 2$ to $n - 1$
    Define neighborhoods $X, X_{-1}, X_{+1}, Y, Y_{-1}, Y_{+1}$ of v_i.
    Calculate E_{cont}, E_{curv}, and E_{image}.
    $E_{total} = \alpha E_{cont} + \beta E_{curv} + \gamma E_{image}$.
    Move v_i to the position of minimum energy.

if v_i moves
$pts\_moved = pts\_moved + 1$
end
end
$v_1 = v_{n-1}$ and $v_n = v_2$ (moving overlapping points)
end

Results of the Head-Finding Algorithm

The head-finding algorithm was tested on pictures of Virginia Tech mathematics students and faculty. The pictures were taken at the Interdisciplinary Center for Applied Mathematics with a uniform color background. The uniformity of the background is extremely important because convergence of the contour is only local. Otherwise, the contour may be tempted to settle on other edges and areas of high image gradient in the background.

Overall, the head-finding algorithm performed well. In most cases, the contour converged to the outline of the top and sides of the head almost perfectly. The region around the neck and chin was more difficult because of the numerous areas of high image gradient. Sometimes the contour would converge to the collar where the shirt and neck meet, sometimes it would stop on the chin, and sometimes a combination of the two. The pictures in Figure 14 demonstrate each case. Note that in each the pictures, the contour was placed in the same place and the same parameters of $\alpha = 1, \beta = 1.8$, and $\gamma = 1.2$ were used. The location of convergence depended on several factors: the initial placement

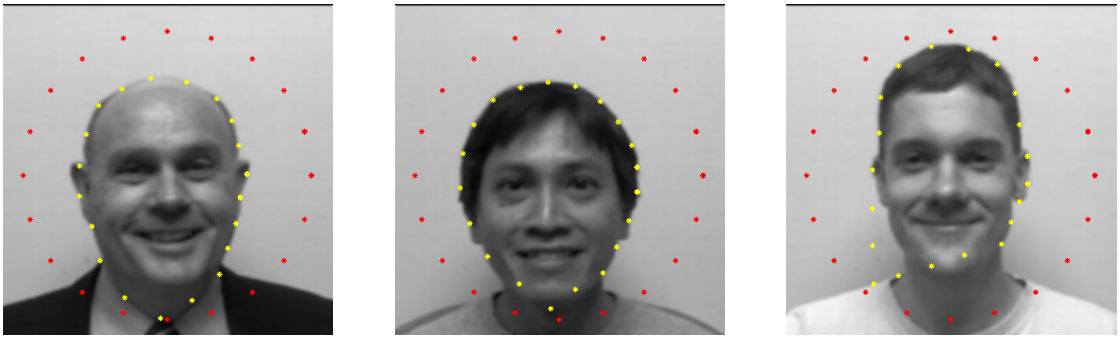

Figure 14: Locations of Convergence: Collar (left), Chin (middle), A Combination (right)

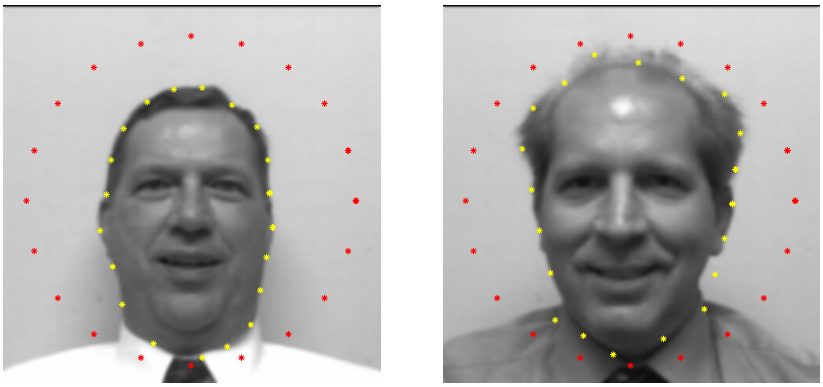

Figure 15: More Examples of Locations of Convergence

of the contour in relation to the head, the magnitude of the curvature parameter β, and the size of the neighborhood around points on the contour. Before further discussion and examples, we briefly mention the parameters and threshholds.

Convergence of the contour occurs when the number of points moved in one iteration is less than the specified *points_moved* threshold. The *points_moved* threshold is fairly flexible, and depended mostly on the number of points on the contour. In our experiments, 2-3 points worked well for convergence of the head-finding algorithm. Between 18 and 22 points on the contour are used. The number of points depends on the size of the image matrix and object being detected. Too many points causes bunching. The parameters we used for the weights of the energy terms are $\alpha = 1$, $1.5 \leq \beta \leq 1.8$, and $\gamma = 1.2$.

Convergence of the head-finding algorithm is local and depends heavily on the initial placement of the contour and its relation to the object. Consider the examples in Figures 16. Notice that in the picture on the left, the contour converged to the collar. In the picture on the right in Figure 16, the initial contour was placed 10 rows up. Observe how the contour converged to the chin. In addition, for heads that were fairly centered, the head-finding algorithm does exceptionally well. When pictures were extremely off-centered, points tended to bunch. See the examples in Figure 17. Using a larger neighborhood seemed to help the points spread out and converge to the edge.

The location of convergence of the contour also depends on β the curvature parameter. Increasing β forced the contour to stay rounded, and discouraged corners from forming. The pictures in Figure 18 show an example of this dependence.

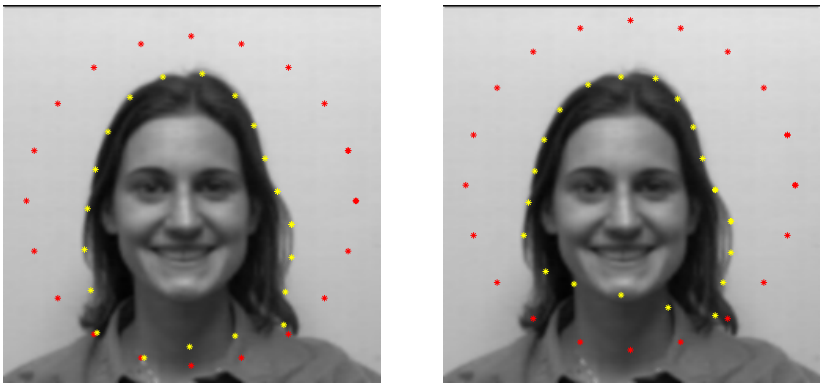

Figure 16: Initial Placement: Converges to collar (left), 10 rows up, converges to chin (right)

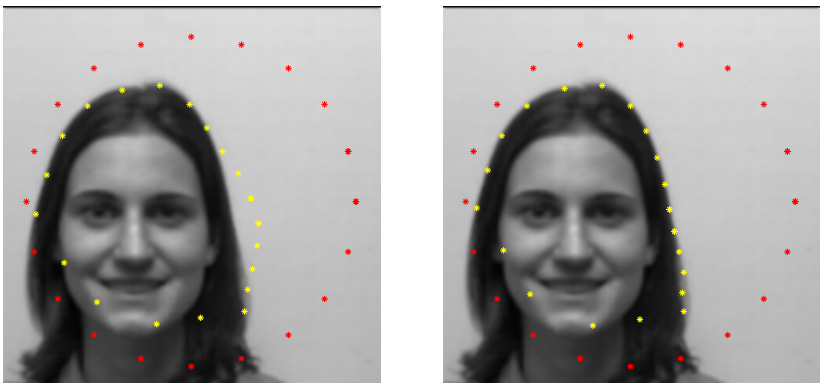

Figure 17: Off-centered Pictures: 5×5 neighborhood (left), 7×7 neighborhood (right)

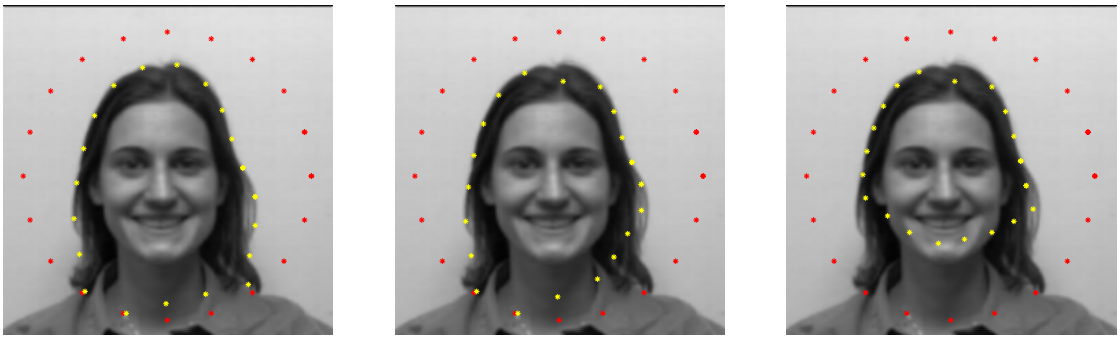

Figure 18: Curvature Parameter: $\beta = 1.8$ (left), $\beta = 2.3$ (middle), $\beta = 2.8$ (right)

The size of the neighborhood around points on the contour also influences where the contour converges. In the examples of the greedy algorithm [11] and the dynamic programming algorithm [1], a 3×3 neighborhood is used. The contours, however, are placed exceptionally close to the object and convergence is achieved in between 3 and 11 iterations. The purpose of the head-finding algorithm, namely, real-time face recognition, prevents the initial contours from being placed as carefully and closely to the head as in these other papers. Thus, it is essential that our algorithm have the ability to move points across far distances. A 3×3 neighborhood around v_i did not prove to be sufficient. Consider the pictures in Figure 19. In the picture on the left, the algorithm

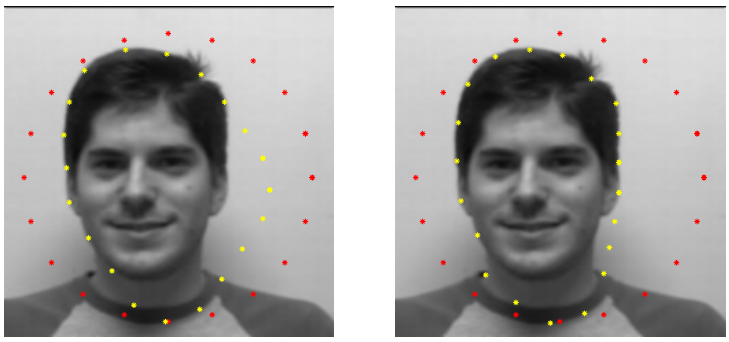

Figure 19: Size of Neighborhood: 3×3 neighborhood (left), 5×5 neighborhood (right)

used a 3×3 neighborhood. Notice how the contour did not make it to the head before converging. It stopped at another local minimum. Now, observe the same picture on the right, now using a 5×5 neighborhood. In this case, the points were able to move all the way to the outline of the head before converging. The results of a 7×7 neighborhood are comparable to 5×5. The 9×9 neighborhood also picked up edges fairly well. In some tests, however, the points on the contour moved to area of high image gradient that were on the face or the hair, indicating that the 9×9 neighborhood may be too large.

6 Conclusions

The rotation of images worked expectionally well on pictures that were shifted horizontally. Vertically off-centered images presented a problem, however. The rotation of the transpose worked best in this case. The rotation recognized the similar in facial geometry in two pictures and always lined up facial features with the original picture. The finer details of the face were usually lost. The coefficient error of the rotated image

with the original image was always smaller than the error of the off-centered picture with the original, which indicated the importance of alignment of the face for accurate respresentation in the eigenface basis. The rotation of images presented one serious potential problem: the possibility of rotating of one person into another. The results in this matter and not totally conclusive due to an insuffiecient eigenface basis for testing, but it appears that the rotation of images in its purest form may not be a good tool for assisting real-time face recognition.

The head-finding algorithm, however, with additional improvements, has the potential to aid real-time face recognition. Before, discussing future work, there are some strengths and weaknesses of the algorithm to point out. First, it finds the top and the sides of the head exceptionally well, but has difficulties with the chin area due to the various edges. It is fast, a *must* for real-time face recognition, taking about half a second to converge. The formulation of the array neighborhoods around the points on the contour makes it easy to compute the energy terms and increase the size of the neighborhood. A larger neighborhood encouraged points that were initially far from the head to move and eventually converge to an edge. Finally, it is flexible and easily allows for the addition of hard constraints and altering the parameters. The main weakness of the head-finding algorithm, which is inherent to this technique of energy-minimizing active contours, is that convergence is local and extremely sensitive to the initial placement of the contour. In addition, a noisy background could cause points to get stuck and stop at some other local minimum far from the outline of the head. Overall, the head-finding algorithm has the potential to improve real-time face recognition.

7 Future Research

The final goal of locating the eyes and aligning them into a standard position has not yet been achieved, but we are one step closer. The exact location of the top and the sides of the head in now known, which gives a much better idea of the position of the eyes. Next, we will attempt to find the eyes using the energy-minimizing active contours. If we had tried to locate the eyes with the active contour algorithms from the beginning, we never would have found them. Convergence is only local and extremely sensitive to the placement of the initial contour. Even now, with a better guess to the position of the eyes, it will be difficult to place new contours around the eyes in such a way to guarantee convergence consistently in every picture. More work is required than simply running the algorithm again. There are too many edges and points of high image gradient in that region. Fortunately, this method of energy-minimizing active contours is flexible. Up to this point, we have not added any hard constraints. In addition, in

the discussion of the greedy algorithm, we saw that different energy terms influence the motion of the contour and the location of convergence. With a closer look at controlling the contour using different energy terms, altering parameters, and with the addition of hard constraints, we hope to develop an eye-finding algorithm that will improve the eigenface method for real-time face recognition.

Bibliography

[1] A.A. Amini, R.C. Jain, and T.E. Weymouth "Using Dynamic Programming for Solving Variational Problems in Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 9, pp. 855-867, Sept. 1990.

[2] D. Bau, III and L.N. Trefethen "Numerical Linear Algebra," *Society of Industial and Applied Mathematics*, Philadelphia, PA, 1997.

[3] I. Borg and P. Groenen "Modern Multidimensional Scaling: Theory and Applications," *Springer-Verlag New York, Inc.*, New York, 1997.

[4] J.W. Demmel "Applied Numerical Linear Algebra," *Society of Industial and Applied Mathematics*, Philadelphia, PA, 1997.

[5] G.H. Golub and C.F. Van Loan "Matrix Computations," third ed. *The Johns Hopkins University Press*, Baltimore, MD, 1996.

[6] M.Kass, D. Terzopoulos, and A. Witkin "Snakes: Active Contour Models," *International Journal of Computer Vision*, Vol. 1, No. 4, pp. 321-331, 1988.

[7] M.Kirby and L. Sirovich "Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 1, pp. 103-108, Jan. 1990.

[8] R. Malladi, J.A. Sethian, and B.C. Vemuri "Shape Modeling with Front Propagation: A Level Set Approach,"Center for Pure and Applied Mathematics, Univ. of California, Berkely and Dept of CIS, Univ. of Florida, Gainsville, 1994.

[9] J.A. Sethian "Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Materials Science " *Cambridge University Press*, Cambridge, MA, 1996.

[10] L. Sirovich "Image Analysis, A Turorial, Part 1. Basics," June 2001.

[11] D.J. Williams and M. Shah "A Fast Algorithm for Active Contours and Curvature Estimation," *CVGIP: Image Understanding*, Vol. 55, No. 1, pp. 14-26, Jan. 1992.

[12] H.S. Wisniewski, "Course Notes: Mathematics of IT," Virginia Tech, Summer 2001.

Vita

Kathleen Ann Bellino is a native of Washington, DC and the second oldest of seven children of Joseph and Pamela Bellino. She received a Bachelor of Science in Mathematics from James Madison University in May 2000. During her four years at James Madison Univeristy, she was a member of the Varsity Women's Gymnastics Team. After graduation from James Madison Univerisity, Kathleen continued her graduate studies in applied mathematics at Virginia Polytechnic Institute and State University. She plans to continue research in applied mathematics, while enjoying the beach and warm weather of San Diego.