Principal Component Analysis for the Approximation of a Fruit as an Ellipse

Sudanthi N.R. Wijewickrema, Andrew P. Papliński
Computer Science and Software Engineering
Monash University, Australia

September 3, 2004

Abstract

Principal component analysis (PCA) has been widely used in fields such as computer vision and image processing since its inception. Conceptually it describes the shape of a cluster of data starting with the direction in which there is most data variability.

We investigate the novel idea of fitting an encapsulating ellipse to an image of a hypothetically ellipsoidal fruit using principal component analysis. This is achieved by representing the fruit as a distribution of data points in 2-D space, identifying the principal components and the variance of the data along the principal components, and determining the ellipse parameters using them.

1 Introduction

Principal component analysis (PCA) is a classical statistical technique which analyses the covariance structure of multivariate data [1].

It is a mathematical way of determining the directions along which the variation of data occur and the corresponding importance of that direction with regard to the set of data points. The first principal component gives the direction in which the variance of the data is maximal, the second principal component is the direction of maximal variance in the direction orthogonal to the first principal component and so on [2].

We investigate the application of principal component analysis to represent the image of an ellipsoidal object such as a fruit, as an ellipse, or in other words as a set of parameters representing the ellipse. This allows for the reduction and simplification of data, making it easier for the higher stages of processing.

Furthermore, we define a measure for calculating the error of fit, and outline a simple technique to determine the goodness of fit. This technique is aimed at applications such as fruit sorting, where resource constraints and speed requirements necessitate the approximation of data.

2 Existing Work

The ellipse being the perspective projection of the circle, has been a very commonly used geometric primitive in the parameter estimation process. Existing work of this nature can broadly be categorized into two types: least squares fitting and clustering.

Least squares fitting techniques focus on finding a set of parameters that minimize some distance measure between the data points and the ellipse. The implicit second order polynomial given in eqn (1) is used to fit the data points to a conic and the constraint $b^2 - 4ac < 0$ is used to represent the conic as an ellipse.

$$F(p, X) = p \cdot X = 0 \ , \quad b^2 - 4ac < 0 \tag{1}$$

where, $X = [x^2\ xy\ y^2\ x\ y\ 1]^T$ and
$p\ = [a\ b\ c\ d\ e\ f]$ is a vector of parameters

The distance measure between data points and the ellipse being minimized may vary depending on the method used. Least squares fitting based on algebraic distances, Euclidean distances and gradient weighted fitting [3] are some of the methods that come under this category. Least squares fitting is a simple method and is often used as a first estimation in other methods of ellipse fitting. Iterative refinement procedures such as Kalman filters [3, 4] and renormalization [3, 5] are used to obtain a more accurate estimation in the case of outliers to which the least squares methods are extremely sensitive. These techniques often increase the amount of computations greatly.

Clustering methods focus on mapping sets of points to the parameter space which is appropriately quantized depending on the application. Hough transform [3] and accumulation methods [6] are examples of this type of technique. These Hough-like techniques have some great advantages, notably high robustness to occlusion and no requirement for pre-segmentation, but they suffer from the great shortcomings of high computational complexity and non-uniqueness of solutions, which can render them unsuitable for real applications [7].

One thing that all these techniques have in common is that they concentrate on fitting an ellipse to a set of points which essentially means that an edge detection algorithm has to be performed on the image before fitting the ellipse. In addition to that, the computational burden of these methods is somewhat high.

In comparison, the method we propose does not require edge detection as a prerequisite to ellipse fitting and needs only a few simple operations to calculate the ellipse parameters. Furthermore, the set of data points can be drastically reduced appropriately in an extremely simple manner without diminishing the accuracy of the outcome.

3 Method of calculation

The estimation of an image of a fruit to an ellipse is done by performing the following basic steps.

1. Acquiring the set of data points
2. Calculating the mean of data (the center of the ellipse)
3. Calculating the covariance matrix
4. Calculating the eigenvalues and eigenvectors of the covariance matrix
5. Approximating the image of the fruit as an ellipse

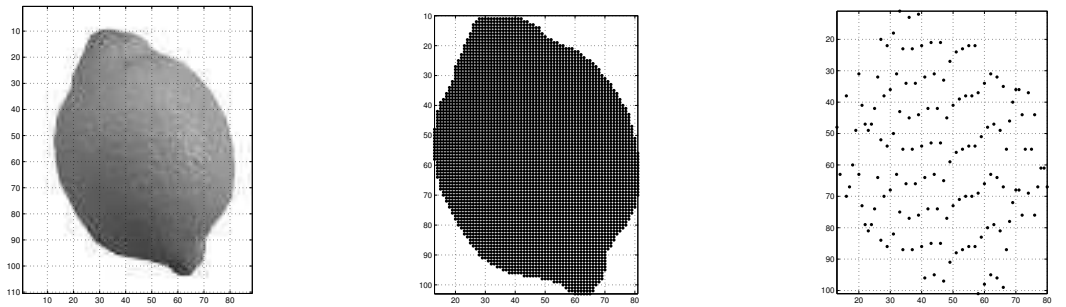
3.1 Acquiring the set of data points

To perform principal component analysis, a set of data points is required to be extracted from the image. As a prerequisite to this, the image is converted into gray scale to simplify the detection of data points.

The assumption that the image is acquired under uniform illumination and that the background of the image is light with respect to the fruit is applied at this point. This assumption is easily satisfied in the type of application this method is aimed at, namely, fruit sorting. Next, basic global thresholding [8] is used to perform the image segmentation, that is, to extract the required set of data points. The brightness threshold may vary depending on the type of fruit being modeled. This could be easily extended to be applied to images obtained under uncontrolled environments by using any suitable segmentation method, e.g., adaptive thresholding techniques [8].

An important point that needs to be highlighted is the decimation of data points performed on the image that results from the thresholding operation. A well chosen small subset of points can be used in further operations instead of the full set, without compromising the accuracy of the outcome of principal component analysis.

Any sensibly balanced method of decimation can be used for this purpose, extracting randomly, a prescribed number of points from the image frame. In our specific implementation we go for computational simplicity selecting every d^{th} pixel from the image that falls inside the segmented image. If d is co-prime with R or C, (the number of rows or columns of the image respectively, depending on the direction of selection: column-wise or row-wise), it should give good random uniform distribution.



(a) Grayscale image (b) Segmented image (c) Decimated image

Figure 1: Example of a fruit

An example of a fruit image is given in Figure 1. Firstly we present an example of a gray scale image (1a), then a segmented image (1b), and finally the data points from the image after decimation (1c).

3.2 Calculating the mean

After the acquisition of the subset of data on which PCA is performed, the mean vector $\mathbf{c} = [c_1 \ c_2]$ should be calculated. It will point to the centre of the ellipse. The first step before calculating the covariance matrix involves the removal of the mean from the data. This results in the data points being centered around the origin as illustrated in Figure 2.

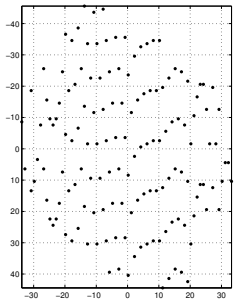


Figure 2: Distribution after removing the mean

3.3 Calculating the covariance matrix

Let the set of data X consist of n m-dimensional vectors

$$X = \begin{bmatrix} \mathbf{x}(1) \\ \vdots \\ \mathbf{x}(n) \end{bmatrix} \quad \text{where} \quad \mathbf{x}(k) = [x_1(k) \dots x_m(k)] \tag{2}$$

Note that in the case of images ($m = 2$) the matrix X is $n \times 2$ and consists of the coordinates of all the points of the decimated image.

The covariance matrix S, of the m-dimensional vectors provides a measure of how strongly correlated their components (variables) are and can be calculated as

$$S = \frac{1}{n-1} \hat{X}^T \cdot \hat{X} \quad \text{where} \quad \hat{X} = X - \mathbf{c} \tag{3}$$

Note that

- $\hat{X}$ represents the data X with the mean $\mathbf{c}$ removed.
- S is a $m \times m$ matrix consisting of the sums of all possible products of all m components of n vectors.
- The matrix S is symmetrical and positive-definite.

Alternately, we can calculate the covariance matrix for $m = 2$ in the following way

$$S = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} \tag{4}$$

where

$$\sigma_{ii} = \frac{1}{n-1}\sum_{k=1}^{n}[x_i(k)]^2, \quad i = 1, 2 \quad \text{and} \quad \sigma_{12} = \sigma_{21} = \frac{1}{n-1}\sum_{k=1}^{n} x_1(k) \cdot x_2(k)$$

3.4 Calculating the eigenvectors and eigenvalues

An $m \times m$ matrix S is said to have an eigenvector v and a corresponding eigenvalue λ if

$$S \cdot v = \lambda \cdot v \tag{5}$$

In other words, if the product of the matrix S and its eigenvector v is calculated, the resulting vector preserves the direction of v and its length will only be changed by a factor λ as illustrated in Figure 3. Any $m \times m$ matrix has a

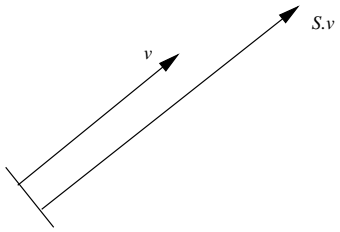


Figure 3: The relationship between a matrix S and its eigenvector v

set of m eigenvalues, some of them possibly identical. Eigenvalues can be real or form pairs of complex conjugate values. The good news is that a positive definite matrix, like the covariance matrix, has only real eigenvalues. Eqn (5) can be written in a form of a matrix decomposition as follows:

$$S \cdot V = V \cdot \Lambda\ , \quad \text{or} \quad S = V \cdot \Lambda \cdot V^{-1} \tag{6}$$

where Λ is a diagonal matrix of eigenvalues, and V is a matrix whose columns are the corresponding eigenvectors.

As the set of eigenvectors or principal components (directions) resulting from this operation are orthogonal to each other and are of unit length, the matrix of eigenvectors is an orthogonal matrix [9]. Hence, its inverse is equal to its transpose as in eqn (7).

$$S = V \cdot \Lambda \cdot V^T \tag{7}$$

For matrices with low dimensionality, eigenvalues can be easily calculated from the characteristic polynomial of the matrix. Eqn (5) can be rewritten in the form:

$$(S - \lambda I)\mathbf{v} = 0 \tag{8}$$

Hence the characteristic polynomial must be zero:

$$\det(S - \lambda I) = 0 \tag{9}$$

From eqn (9), we obtain the following quadratic equation, solving which we can obtain the two eigenvalues, λ_1 and λ_2 as shown in eqn (11).

$$\begin{vmatrix} (\sigma_{11}-\lambda) & \sigma_{12} \\ \sigma_{12} & (\sigma_{22}-\lambda) \end{vmatrix} = 0 \quad \longrightarrow \quad \lambda^2 - \lambda(\sigma_{11}+\sigma_{22}) + (\sigma_{11}\sigma_{22} - \sigma_{12}^2) = 0 \tag{10}$$

Hence that eigenvalues are:

$$\lambda_{1,2} = \frac{1}{2}\left(\sigma_{11}+\sigma_{22} \pm \sqrt{(\sigma_{11}-\sigma_{22})^2 + 4\sigma_{12}^2}\right) \tag{11}$$

The eigenvectors of the matrix can then be calculated from eqn (8). Taking into account that the two eigenvectors are orthogonal, it can be seen that only two independent scalars, say v_1, v_2 are needed to characterize the two eigenvectors, so that eqn (8) can be written as:

$$\begin{bmatrix} \sigma_{11}-\lambda_1 & \sigma_{12} \\ \sigma_{12} & \sigma_{22}-\lambda_2 \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 \\ v_2 & -v_1 \end{bmatrix} = 0 \tag{12}$$

From eqn (12) we get one independent equation:

$$v_1(\sigma_{11}-\lambda_1) + v_2\sigma_{12} = 0 \tag{13}$$

Additionally the eigenvectors should be normalized so that their norms are unity. This gives us the equation:

$$v_1^2 + v_2^2 = 1 \tag{14}$$

Eqns (13) and (14) uniquely specify the eigenvectors:

$$v_1 = \frac{\sigma_{12}}{\sqrt{(\lambda_1-\sigma_{11})^2+\sigma_{12}^2}}, \quad v_2 = \frac{\lambda_1-\sigma_{11}}{\sqrt{(\lambda_1-\sigma_{11})^2+\sigma_{12}^2}} \tag{15}$$

Jacobi rotations

For matrices with higher dimensionality, the above discussed method of calculation may not be very suitable and in such instances methods such as Jacobi Rotations or Givens Transformations [10] should be used.

To explain the Jacobi rotations let us write eqn (7) in the following form:

$$\Lambda = V^T \cdot S \cdot V \tag{16}$$

Jacobi rotations aim at forming this diagonal matrix of eigenvalues Λ, by performing rotations iteratively on the symmetric (covariance) matrix S.

At each iteration a plane rotation is performed using the rotation matrix, P_{pq}, as shown in eqn (17), where an element (p, q) along with the element (q, p) is converted to zero.

$$S' = P_{pq}^T \cdot S \cdot P_{pq}, \quad p \neq q \tag{17}$$

where

$$P_{pq} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

Here all the diagonal elements are unity except for the two elements c in rows (and columns) p and q. All off-diagonal elements are zero except the two elements s and $-s$.

where $c = \cos\phi$ and $s = \sin\phi$

therefore $c^2 + s^2 = 1$.

When the diagonal matrix is formed at the end of the iterations, the following relationship holds true, indicating that the product of the rotation matrices gives the matrix of eigenvectors as in eqn (16).

$$\Lambda = (P_n^T \cdots P_2^T \cdot P_1^T) \cdot S \cdot (P_1 \cdot P_2 \cdots P_n) \quad \longrightarrow \quad V = P_1 \cdot P_2 \cdots P_n \tag{18}$$

Jacobi rotations for the 2-D case

For our purpose, only one rotation needs to be performed on the 2×2 symmetric covariance matrix. The rotation as specified in eqn (19) gives the final result as shown in eqn (20).

$$P_{12} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{19}$$

$$\Lambda = P_{12}^T \,.S.\, P_{12}, \quad V = P_{12} \tag{20}$$

$$\Lambda = \begin{bmatrix} c^2\sigma_{11} - 2cs\sigma_{12} + s^2\sigma_{22} & sc(\sigma_{11} - \sigma_{22}) + \sigma_{12}(c^2 - s^2) \\ sc(\sigma_{11} - \sigma_{22}) + \sigma_{12}(c^2 - s^2) & s^2\sigma_{11} + 2cs\sigma_{12} + c^2\sigma_{22} \end{bmatrix} \tag{21}$$

Taking into account that the off-diagonal elements of Λ are zero we obtain the following equation:

$$\frac{c^2 - s^2}{sc} = \frac{\sigma_{22} - \sigma_{11}}{\sigma_{12}}$$

We define θ as follows,

$$\theta = \cot 2\phi = \frac{c^2 - s^2}{2sc} = \frac{\sigma_{22} - \sigma_{11}}{2\sigma_{12}} \tag{22}$$

If we let $t = \tan\phi = s/c$,

$$t^2 + 2t\theta - 1 = 0$$

The smaller root of this equation corresponds to the rotation angle less than $\pi/4$ in magnitude giving the most stable reduction. Using the form of the quadratic formula with the discriminant in the denominator we get the smaller root as:

$$t = \frac{sgn(\theta)}{|\theta| + \sqrt{\theta^2 + 1}} \tag{23}$$

It now follows that,

$$c = \frac{1}{\sqrt{t^2 + 1}} \quad \text{and} \quad s = tc \tag{24}$$

Substituting s and c in eqns (20) and (21) we get the eigenvectors and the eigenvalues of the covariance matrix.

Comparison of methods

Using eqns (11) and (15) we can obtain the eigenvectors and eigenvalues of the 2×2 covariance matrix using basic concepts. These equations indicate that the number of calculations required for the solving of the eigensystem is not very heavy with respect to the amount of processing required. The two square root operations that are required are the only cost intensive operations that are necessary to be performed.

Similarly eqns (22), (23) and (24) show that if the eigensystem is solved using Jacobi rotations, it also requires two square root operations indicating that processing time and cost for both these methods do not differ by much making either of them suitable for the calculation of eigenvectors and eigenvalues of a 2×2 matrix.

3.5 Approximating the image of the fruit as an ellipse

The mean of data, c_1, c_2, the eigenvalues $\lambda_{1,2}$, and the direction of the first eigenvalue, ϕ, uniquely define the ellipse [11]. This equation can be conveniently written in a complex-number parameterized form as follows:

$$x_1 + jx_2 = (a_1 \cos t + ja_2 \sin t)e^{j\phi} + (c_1 + jc_2) \tag{25}$$

where

$$a_i = 2\sqrt{\lambda_i}$$

In Figure 4 such an ellipse is fitted around a fruit.

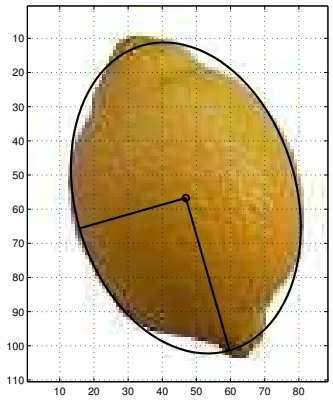


Figure 4: The principal components and the estimated ellipse

4 Experimental Results

The experimental results of the above discussed ellipse fitting algorithm performed on different types of fruit is shown in Figure 5.

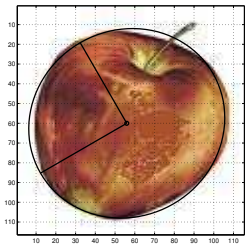
It should be noted here that depending on the colour of the fruit and the colour of the background, a suitable threshold for segmentation should be chosen to ensure accurate results.

Error of fit

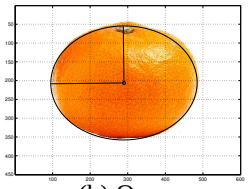
To determine how good a certain ellipse fitting algorithm is, we need to calculate the error of fit. For this, many error calculation methods [12] can be used. To get a simple yet accurate measure of error, we calculate the algebraic distance of the edge points of the image of the fruit to the estimated ellipse.

The algebraic distance D, of a point X_i, to a curve $F(p, X) = 0$, can be defined as in eqn (26).

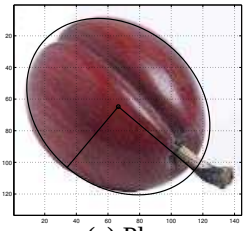
$$D = F(p, X_i) \tag{26}$$



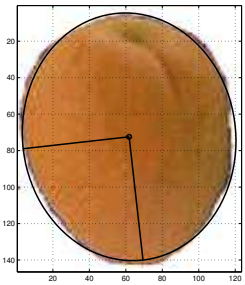
(a) Apple



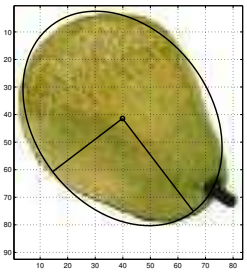
(b) Orange



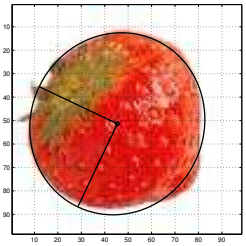
(c) Plum



(d) Apricot



(e) Pear



(f) Strawberry

Figure 5: Examples of some images of fruit approximated as ellipses

where, $X = [x^2\ xy\ y^2\ x\ y\ 1]^T$ and
$p\ = [a\ b\ c\ d\ e\ f]$

We define an error coefficient e, the normalized sum of squares of algebraic distances as given in eqn (27).

$$e = \frac{1}{n}\sum_{i=1}^{n} D_i^2 \tag{27}$$

where, n is the total number of edge points

The algebraic distance illustrates the deviation of the estimated ellipse from an edge point. The normalization is done so as to remove the dependency that the sum of squares has on the number of edge points used in the calculation. This enables the error measure e, to be used for comparative purposes.

It can be observed that the error of fit of the ellipse changes with the value of the decimation constant d. To generalize the selection of d for each and every image, we define a value G, so that an optimal set of points can be selected, that minimizes the processing and produces an acceptable error of fit. Figure 6 shows how the optimal value, g is defined for a line of unit length and the relationship in eqn (28) holds true in such a situation.

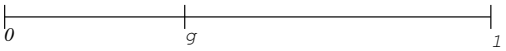


Figure 6: The optimal value g, for a line of unit length

$$\frac{g}{1-g} = \frac{1-g}{1} \tag{28}$$

Solving for g, we get:

$$g = \frac{3 \pm \sqrt{5}}{2},$$

from which we choose the smaller value to ensure that enough points are selected. Therefore, the optimal value for a line of unit length is defined as given in eqn (29).

$$g = \frac{3 - \sqrt{5}}{2} \tag{29}$$

The optimal decimation constant G, for a discrete set of image pixels, where R is the number of rows of the image (for a column-wise decimation), can be obtained from this as shown in eqn (30).

$$G = \text{round}(R \times \frac{3 - \sqrt{5}}{2}) \tag{30}$$

Figure 7 shows how the error coefficient e, changes with the decimation constant d and the optimal value G, defined to optimize results for the image shown in Figure 1.

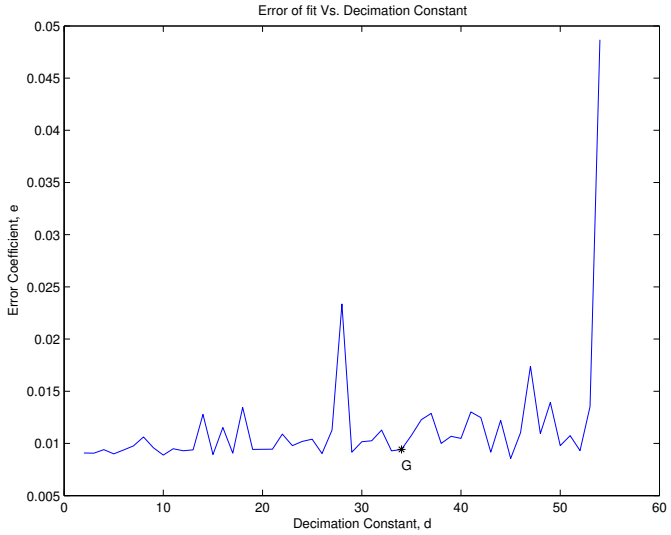


Figure 7: Error of fit Vs. Decimation constant

Goodness of fit

The goodness of fit of a certain image to an ellipse is a measure that needs to be calculated, to discover whether it is feasible to approximate that particular image as an ellipse or not.

We use the error measure e, to determine if the fit is acceptable or not. We define a threshold T, for the error of fit, and it is accepted that any error measure appearing below that threshold indicates that the approximation is valid.

The threshold T, is determined experimentally where images of different shapes are fit into an ellipse and the error of fit is calculated (for one chosen value of d, in this case, G). Then the elliptic images and the non-elliptic images can be separated using the threshold depending on the values obtained for the error of fit. Figure 8 shows one such set of shapes and the resulting error coefficients.

The threshold could then be used to determine if a certain fit is valid or not.

Conclusion

The application of principal component analysis to estimate the image of a fruit as an ellipse is investigated as a technique to be used in fruit sorting applications.

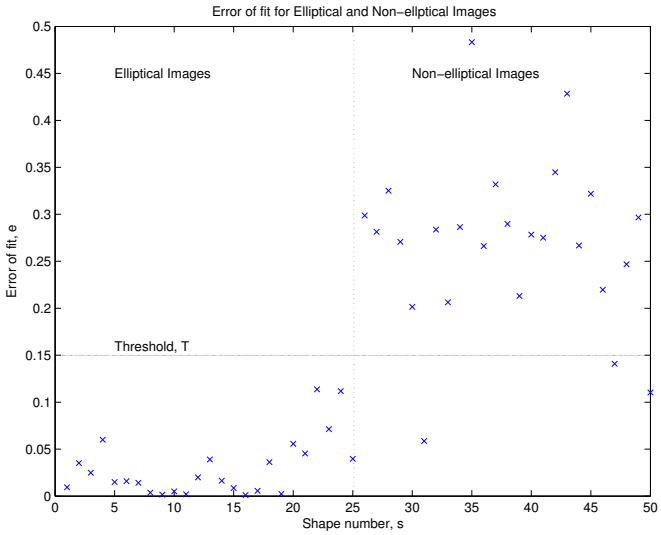


Figure 8: Error of fit for different shapes

The processing here has to be done in real-time, and hence great emphasis is placed on speed. The algorithm and techniques used in this process are therefore aimed at simplifying the calculations and maximizing efficiency. For instance basic global thresholding is used in place of a more sophisticated segmentation technique for the acquisition of data. This does not compromise the accuracy of the data as the images are obtained under controlled conditions. Decimation of data and the use of basic concepts or Jacobi rotations to solve the eigensystem instead of a more complex algorithm are also aimed at achieving higher speed.

The use of principal component analysis in the representation of an image of a fruit as an ellipse is concluded therefore to be an efficient and suitable method to be used in fruit sorting applications in real-time and could be easily adapted for other similar applications.

References

[1] H. Hotelling, “Analysis of a complex of statistical variables into principal components,” *Journal of Educational Psychology*, vol. 24, 1933.

[2] A. Weingessel and K. Hornik, “Local PCA algorithms,” *IEEE Transactions on neural Networks*, vol. 11, no. 6, November 2000.

[3] Z. Zhang, “Parameter estimation techniques: A tutorial with application to conic fitting,” *Image and Vision Computing Journal*, vol. 15, no. 1, pp. 59–76, 1997.

[4] J. Porril, “Fitting ellipses and predicting confidence envelopes using a bias corrected kalman filter,” *Image and Vision Computing*, vol. 8, no. 1, pp. 37–41, February 1990.

[5] K. Khanatani, “Statistical bias of conic fitting and renormalization,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 3, pp. 320–326, March 1997.

[6] P. Rosin, “Ellipse fitting by accumulating five-point fits,” *Pattern Recognition Letters*, vol. 14, no. 8, pp. 661–669, 1993.

[7] A. W. Fitzgibbon, M. Pilu, and R. B. Fisher, “Direct least squares fitting,” *Tern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 476–480, May 1999.

[8] R. C. Gonzales and R. E. Woods, *Digital image Processing*. Prentice-Hall, 2002.

[9] K. Heuvers, J. Kuitsi, W. Francis, G. Ortner, D. Moak, and D. Lockhar, *Linear Algebra for Calculus*. Brooks-Cole, 1991.

[10] W. H. Press, B. P. Flannery, and W. T. Teukolsky, Saul A. Vetterling, *Numerical Recipes in C: The art of scientific computing*. Cambridge University Press, 1988–1992.

[11] I. Jolliffe, *Principal Component Analysis*. Springer, 2002.

[12] P. L. Rosin, "Analysing error of fit functions for ellipses," *British Machine Vision Conference*, 1996.

Appendix: Implementation in Matlab

% **Clear all variables and figures**
clear all;
close all;

% **Read the image**
$fruitImage = imread('Images/lemon.jpg');$
$grayImage = rgb2gray(fruitImage);$

% **Define Constants**
$LEVEL = graythresh(grayImage);$
$THRESHOLD = 255 * LEVEL;$

$imageSize = size(grayImage);$
$DEC\_CONST = round(imageSize(2) * (3 - sqrt(5))/2);;$

% **Calculate the principal components**
$[dataMean, param, eigVec] = findPC(grayImage, THRESHOLD, DEC\_CONST);$

% **Draw the image and principal components**
$drawPC(fruitImage, dataMean, param, eigVec);$

% **Find the Error of Fit**
$normSumSquare = findError(grayImage, LEVEL, eigVec, param, dataMean);$

% **Function *findPC* - calculates the ellipse parameters**
function $[dataMean, param, eigVec] = findPC(grayImage, THRESHOLD, DEC\_CONST)$
$CONST = 2;$
$noOfRows = size(grayImage, 1);$
$colImage = grayImage(:);$

% **Perform a simple segmentation**
$[Z(:,1), Z(:,2)] = find(colImage < THRESHOLD);$

% **Decimate the data**
$[I, J] = find(mod(Z(:,1), DEC\_CONST) == 1);$
$A = Z(:,1);$
$A = A(I);$

% **Convert back to a 2D matrix with the original indices**
$Y(:,1) = ceil(A./noOfRows);$
$B = mod(A, noOfRows);$
$B(B == 0) = noOfRows;$
$Y(:,2) = B;$
$nY = size(Y,1);$

% **Calculate the mean**
$dataMean = mean(Y,1);$

% **Remove the mean from the data**
$Y = Y - dataMean(ones(nY,1),:);$

% **Calculate the covariance matrix**
$covMatrix = (Y' * Y)./nY;$

% **Find the axes of the ellipse using one of the methods:**
% **findEig(covMatrix), findEigMatlab(covMatrix) or findEigJacobi(covMatrix)**
$[eigVec \;\; eigVal] = findEig(covMatrix);$
$param = CONST * sqrt(eigVal);$

% **Function *drawPC* - draws the approximated ellipse and the image**
function $drawPC(fruitImage, dataMean, param, eigVec)$
% **Calculate the angular parameters**
$k = 100;$
$fi = 2 * pi * (0:k)/k;$

$ell = param(1) * cos(fi) + j * param(2) * sin(fi);$
$ell = ell * (eigVec(1) + j * eigVec(2)) + (dataMean(1) + j * dataMean(2));$

% **Draw the ellipse and the image**
image(fruitImage) , axis('image'), hold on
$plot1 = plot(ell,'k'); gridon;$
$plot2 = plot(dataMean(1), dataMean(2),'ko');$
$plot3 = plot([0 \;\; param(1) * eigVec(1,1)] + dataMean(1), [0 \;\; param(1) * eigVec(2,1)] + dataMean(2),'k');$
$plot4 = plot([0 \;\; param(2) * eigVec(1,2)] + dataMean(1), [0 \;\; param(2) * eigVec(2,2)] + dataMean(2),'k');$

% **Set line width of the plots**
set(plot1,'LineWidth',2), set(plot2,'LineWidth',2), set(plot3,'LineWidth',2), set(plot4,'LineWidth',2);

hold off;

% **Function *findEig* - solves the eigensystem using basic concepts**
function$[eigVec, eigVal] = findEig(covMatrix)$

% **Calculate the eigenvalues**
$delta = sqrt((covMatrix(1,1) - covMatrix(2,2))\hat{}2 + 4 * covMatrix(1,2)\hat{}2);$
$eigVal = [(covMatrix(1,1) + covMatrix(2,2) + delta)/2; (covMatrix(1,1) + covMatrix(2,2) - delta)/2];$

% **Calculate the eigenvectors**
$omega = sqrt((eigVal(1) - covMatrix(1,1))\hat{}2 + covMatrix(1,2)\hat{}2);$
$dirn1 = covMatrix(1,2)/omega;$
$dirn2 = (eigVal(1) - covMatrix(1,1))/omega;$
$eigVec = [dirn1, -dirn2; dirn2, dirn1];$

% **Function *findEigJacobi* - solves the eigensystem using Jacobi rotations**
function $[eigVec, eigVal] = findEigJacobi(covMatrix)$

% **Calculate the eigenvectors**
$theta = (covMatrix(2,2) - covMatrix(1,1))/(2 * covMatrix(1,2));$
$t = sign(theta)/(abs(theta) + sqrt(theta\hat{}2 + 1));$
$c = 1/(sqrt(t\hat{}2 + 1));$
$s = c * t;$

$eigVec = [c, s; s, -c];$
$eigVec = flipud(eigVec);$

% **Calculate the eigenvalues**
$eigVal = [covMatrix(1,1) - t * covMatrix(1,2); covMatrix(2,2) + t * covMatrix(1,2)];$
$eigVal = flipud(eigVal);$

% **Function *findEigMatlab* - solves the eigensystem using Matlab functions**
function $[eigVec, eigVal] = findEigMatlab(covMatrix)$

$[eigVec \;\; eigVal] = eig(covMatrix);$
$eigVal = diag(eigVal);$

$eigVec = fliplr(eigVec);$
$eigVal = flipud(eigVal);$

% **Function *[normSumOfSquares] = findError* - calculates the error of fit**
function $findError(grayImage, LEVEL, eigVec, param, dataMean)$

% **Segmet the image and find the edges**
$binaryImage = im2bw(grayImage, LEVEL);$
$binaryImage = bwmorph(binaryImage,' fill');$
$edgeImage = edge(binaryImage,' log');$
$[y \; x] = find(edgeImage);$

% **Find the rotation angle**
$theta = atan(eigVec(1,2)/eigVec(1,1));$
$c = cos(theta);$
$s = sin(theta);$

% **Center the edge points around the origin**

$x = x - dataMean(1);$
$y = y - dataMean(2);$

$nX = size(x, 1);$

% Rotate the edge points to be aligned with the x-axis
$X = (c * x - s * y);$
$Y = (s * x + c * y);$

$diff = (X.\hat{}2)/param(1)\hat{}2 + (Y.\hat{}2)/param(2)\hat{}2 - 1;$
$normSumOfSquares = sum(diff' * diff)./nX;$