Principal Component Analysis
Computer Vision, Lab Exercise 3
Andreas van Cranenburgh (0440949), Willem Bressers (5687063)
May 21, 2008

Abstract

This report discusses the use of Principal Component Analysis to estimate the position of an image using a training set.

Contents

List of Figures

1 Principal Component Analysis

The Principal Component Analysis (PCA) can be used to reduce the dimensionality of data. Without PCA, our data would have as many dimensions as pixels in the data set, which would be intractable for large data sets.

The PCA starts by calculating the the mean image of all the images in the supplied dataset. The mean is calculated by:

$$\vec{x} = \frac{1}{n}\sum_{i=1}^{n}\vec{x_i} \tag{1}$$

where the dataset contains n images. Each $\vec{x_i}$ represents an image in the dataset as a column vector, which all together are summed and divided by n. Now $\vec{x}$ is the mean vector of all images. Next the mean vector is subtracted from each image vector:

$$X_i = (\vec{x_i} - \vec{x}) \tag{2}$$

so that the mean of the dataset becomes 0. Because the images have now been 'normalized' for values that occur in every image, the borders around the images in the data set should no longer influence the result.

1.1 Covariance matrix

Next it is possible to calculate the Covariance matrix C. The Covariance matrix is a symmetric matrix of all possible combinations of different image vectors. The diagonal terms on the matrix represent the *variance* of the vectors. The non-diagonal terms on the matrix represent the *covariance* between the vectors.

$$C = \frac{1}{n-1}X_i X_i^T \tag{3}$$

We used the equation:

$$C = \frac{1}{n-1} X_i^T X_i \tag{4}$$

because we have a dataset of 300 images that are $(112 * 150)$ big which results in C being a (300×300) matrix. Using the former formula the covariance matrix would be (16800×16800), which is usually intractable. We have to take this into account when defining the actual components. The term $\frac{1}{n-1}$ is used because it is an unbiased estimation for small n values, whereas $\frac{1}{n}$ is a biased estimation.

1.2 Eigenvectors and Eigenvalues

The goal of the PCA is to find the Principal Components. To find the Principal Components it is necessary to first find the Eigenvectors. The Eigenvectors can be found by rewriting the covariance matrix:

$$\begin{aligned} C = \frac{1}{n-1} X_i X_i^T = \\ \frac{1}{n-1} (EY)(EY)^T = \\ \frac{1}{n-1} EYY^T E^T = \\ \frac{1}{n-1} E(YY^T) E^T = \\ \frac{1}{n-1} EDE^T \end{aligned} \tag{5}$$

E is a matrix of Eigenvectors and D is a diagonal matrix with Eigenvalues. Now that the Eigenvalues are known it is possible to find the Principal Components.

$$PC = EX_i \tag{6}$$

Because we used a different covariance matrix we have to rewrite the equation:

$$PC = (E^T X_i^T)^T \tag{7}$$

PC is a matrix of the components of the images. Figure 1 shows the first 9 Principal Components of our dataset.

Because the Principal Components are based on the Eigenvectors, it is important to keep in mind that the largest Principal Components is the first one. Figure 2 shows the first 50 Principal components. We want to find the best number of Principal Components without overfitting. Based on this graph 20 components would seem a reasonable number, since the graph shows the components to slowly become negligible after that.

2 Positioning

After having done the Principal Component Analysis we can now try to compare an arbitrary image to the images from the training set. The image that best matches it provides an indication of where it was made.

Positioning might appear trivial with GPS being so ubiquitous nowadays, but consider indoor locations with poor reception, or even space and planetary explorations, where GPS is unavailable.

2.1 Reduced representation

Using the PCA just performed we can create a reduced representation of all images in the training set. With n images, for all i in n the reduced representation is obtained by:

$$g_i = U^T \cdot x_i \tag{8}$$

2.2 Nearest neighbor

Now we can turn to an algorithm for approximating the location of an image. We will first convert the image to the reduced representation as above. Then we estimate the 'distance' between that and each image of the training set (or inversely their similarity) thus[1]:

$$dist_i = norm(g_i - (U^T \cdot image)) \tag{9}$$

Where *image* is a column vector with the mean calculated before subtracted from it. After obtaining the distances between the image and each of the training set images, the one closest to it provides the estimated position of the image.

2.3 Evaluation

This method can be evaluated by comparing the estamited positions with their actual positions, as given in the test set. Once again we calculate the distance (but now of real world coordinates and not of image data), between the estimated and actual positions:

$$dist_i = norm(est_i - pos_i) \tag{10}$$

Running this on all images of the test set enables us to see how well the algorithm performs. For our purposes we will consider a distance of less than 150 units to be a correct estimate. This is because the images in the data set have been taken with about 50 units in between.

Using the first 20 principal components the following score is obtained:

```
>> testPositioning(images)
[...]
correct =

   174

wrong =

    76

Elapsed time is 1.325647 seconds.
>>
```

Experimentally we discovered that for a number of components between 30 and 44 the optimal score of 175 correct can be obtained, after which the accuracy goes to 174 again. This shows that 20 is a good tradeoff in terms of speed versus accuracy.

When the number is decreased the accuracy decreases gracefully, eg. 164 correct with only 10 components.

2.4 Without PCA

As a further benchmark, we tried the positioning without the use of PCA, and instead comparing every picture directly:

```
correct =

   187

wrong =

    63

Elapsed time is 14.043909 seconds.
```

[1] For completeness, $norm(X) = \sqrt{\sum \vec{x}^2}$ with x being a column vector, for our purposes

2.5 PCA Using the SVD

Instead of using the function `eigs` to calculate eigenvectors, it is also possible to use the `svd`. For comparison, we implemented this in `PCAsvd.m`.

```
correct =

   178

wrong =

    77

Elapsed time is 3.385796 seconds.
```

This score was obtained using 108 principal components. A value arrived at by repeated guessing.

2.6 Even more precision

To further improve the accuracy of this method it might be useful to take into account that the images are omnidirectional. In fact the image is a flat representation of a hemisphere, so perhaps the theory of manifolds can help in this regard. Comparing the distance between two images in a way that is rotationally invariant might improve the accuracy, but we have not pursued this yet.

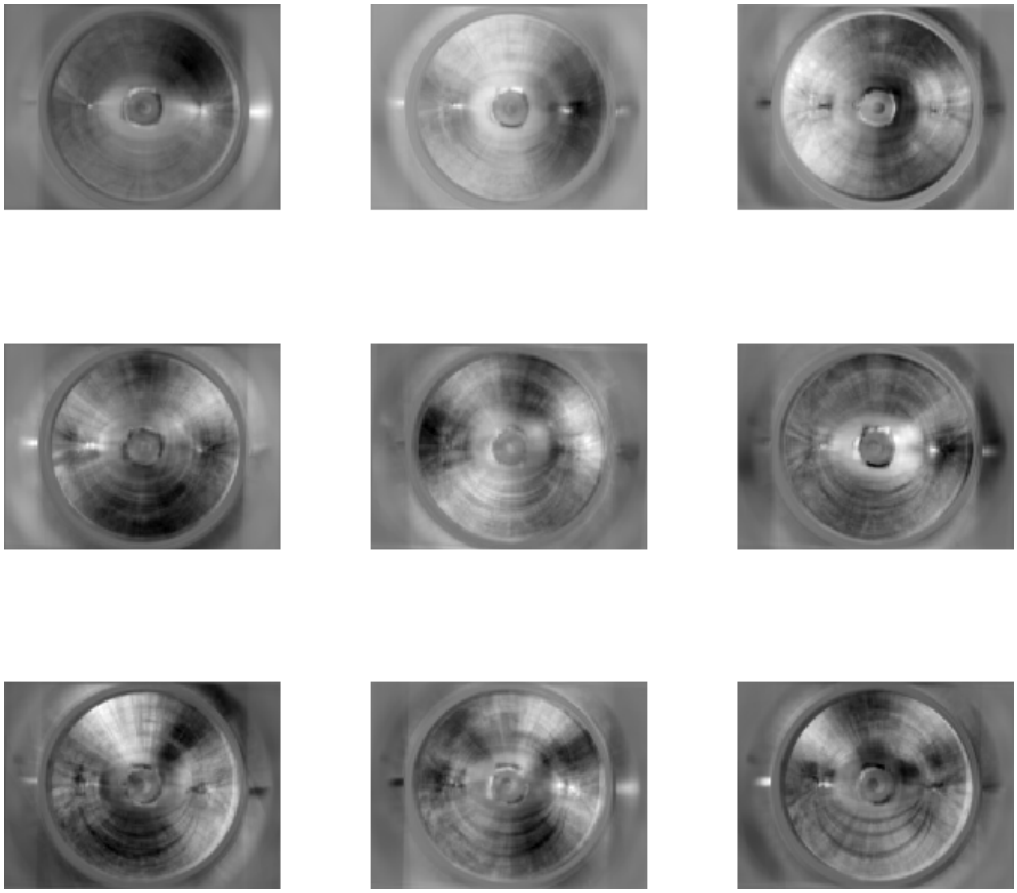

Figure 1: The first 9 Principal Components

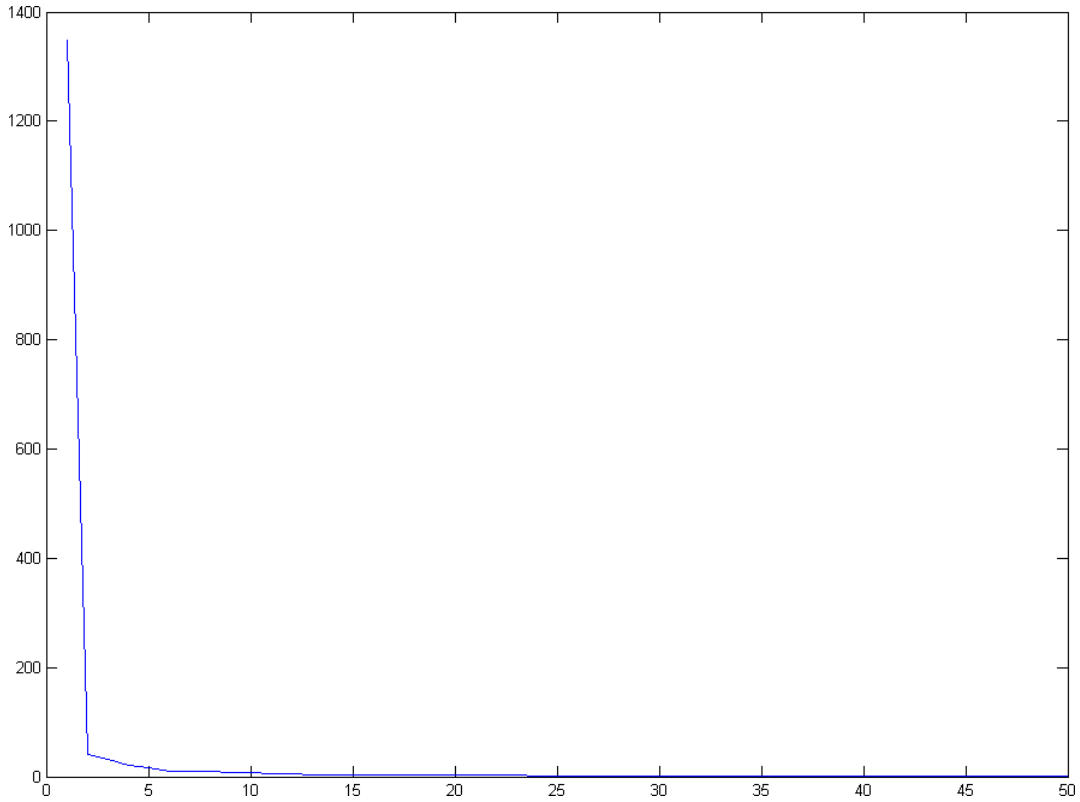

Figure 2: Graph of the first 50 Principal Components