# EEE3032 – Computer Vision and Pattern Recognition

Lab Worksheet 2 – Principal Component Analysis (PCA)

Miroslaw Bober (m.bober@surrey.ac.uk)

**Task and Learning Objective:** In today's lab you will experiment with PCA on synthetic and real data. For the latter, you will apply PCA to locate a distinctively coloured object in an image. Through this lab you will gain experience in putting into practice the theory of PCA covered in lectures.

**Resources:** The exercises will be performed using Matlab. Download the Matlab code for the labs, if you have not done so before, and unzip the code into a separate folder. Ensure the code folder is in the Matlab search path (File -> Set Path.. -> Add with subfolders).

## Ex1: Finding Modes of Variation in Data *(10 minutes)*

*In this exercise you will use a function supplied by your lecturer to perform PCA on a set of 3D data points. You will examine the eigenvectors and eigenvalues and see how they result from the distribution of the data.*

Create a distribution of 5000 random 2D points, with 'x' and 'y' coordinates in the range [0,1].

```
>> pt = rand(2,5000);
```

Plot them.

```
>> plot(pt(1,:),pt(2,:),'bx');
```

Turn these into 3D points by adding a 'z' coordinate, set to zero for all points.

```
>> pt(3,:)=0;
```

The points in pt are now 3D points, all lying along the plane z=0. Plot this in 3D to check:-

```
>> plot3(pt(1,:), pt(2,:), pt(3,:), 'rx');
```

Multiply the y coordinate of the points by 5.

```
>> pt(2,:)=pt(2,:) * 5;
```

The points now lie range from x=[0,1], y=[0,5], and z=0.

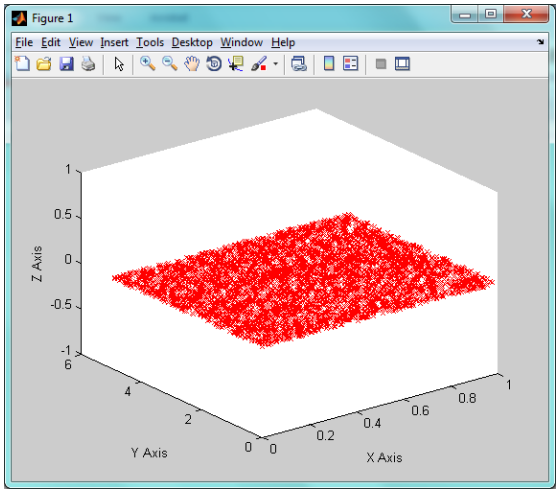
Repeat the "plot3" step above to check. i.e.

```
>> plot3(pt(1,:), pt(2,:), pt(3,:), 'rx');
```

You can label the axes using the xlabel, ylabel and zlabel commands. For example:

```
>> xlabel ('X axis');
>> ylabel ('Y axis');
>> zlabel ('Z axis');
```

You should end up with a plot of your 3D raw data that resembles this:-




**We will now build an Eigenmodel from this 3D data, and use it to determine the modes of variation in the data.**

The eigenmodel will comprise a mean, 3 eigenvectors and 3 corresponding eigenvalues (because the data has 3 dimensions).

**Predictions:**

- We predict the direction of greatest variation in the data to point along the y axis; i.e. the principal eigenvector (the eigenvector with the largest eigenvalue) to point along the y axis because these values are of range [0,5].
- We would predict the eigenvector with the second largest eigenvalue would point along the x axis, as these points are of range [0,1].
- Would would expect the third eigenvector to have a near-zero eigenvalue, because all points have z=0

**Experiment (check the predictions):**

Build the eigenmodel using the Eigen_Build function supplied in the lab code:

```
>> e = Eigen_Build(pt)

e =    N: 5000
 D:       3
      org: [3x1 double]
      vct: [3x3 double]
      val: [3x1 double]
```

Here N is the number of observations used to build the model (5000), the dimension of the observations (3) and the fields 'org', 'vct' and 'val' are the mean, eigenvectors and eigenvalues respectively of the resulting eigenmodel.

Examine the eigenvalues. Due to the random generation of the points, your values will differ slightly:

```
>> e.val
ans = 2.0683
 0.0830
           0
```

Now look at the eigenvectors corresponding to these. Remember each column of 'vct' is an eigenvector.

```
>> e.vct
        ans =
          -0.0013        -1.0000         0
          -1.0000         0.0013         0
           0              0              1.0000
```

Again, your values will vary. However you should see that the first eigenvector is roughly $[0\ 1\ 0]^T$ i.e. pointing along the y axis, the second along the x axis, and the third along the z axis. This is just as we predicted.

Clearly the first eigenvalue is much larger than the second. This is because the spread of points is 5 times larger along the y axis (the first i.e. principal eigenvector) versus the second (the x axis). That is, the standard deviation is 5 times as large. The variance (square of the standard deviation) should thus be 25 times as large. Recall the eigenvalues encode the variance along each eigenvector. Divide the first eigenvalue by the second; you should observe a factor of approximately 25.

## Ex2: Repeating Ex1 - computing the Eigenmodel manually - *(15 minutes)*

*In this exercise you will repeat the analysis of Ex1, but working out the PCA yourself rather than using the EigenBuild code supplied by your lecturer.*

First, compute the mean of the 3D points in pt. Recall this is done by averaging the x, y, and z dimensions of the points. You can do this quickly (take care with the apostrophes) via:

```
>> org=mean(pt')';
```

Or you can use:

```
>> x=sum(pt(1,:))/5000;
>> y=sum(pt(2,:))/5000;
>> z=sum(pt(3,:))/5000;
>> org = [x ; y ; z];
```

Now subtract the mean from the points in pt:

```
>> ptsub = pt - repmat(org,1,5000);
```

Note we used repmat to repeat the 'org' column 5000 times to create a matrix the same size as pt, suitable for the subtraction. Matlab would not have accepted ptsub=pt-org due to a matrix size mismatch.

Now compute the 3x3 covariance matrix from ptsub:

```
>> C = (ptsub*ptsub') ./ 5000
```

Finally, decompose the covariance matrix into its eigenvector and eigenvalues:

```
>> [vct val] = eig(C);
```

Examine org, vct, val and compare them to your previously computed e.org, e.vct, and e.val.
Some notes:

1. the eigenvalues are presented as a diagonal matrix rather than a column.

2. the eigenvalues and their respective eigenvectors may be in a different order. The columns of vct map to columns in val.

3. the eigenvectors might point the opposite way – this is an ambiguity in the solution for the eigen-decomposition. It does not matter; the eigenvectors point along the direction of variation (which is sign independent) and the eigenvalues simply specify the variance in that direction of variation.

## Ex.3: Distance from an eigenmodel *(15 minutes)*

*In this exercise we will build another Eigenmodel from 3D data. Rather than being random these 3D data will comprise red, green and blue (RGB) pixel values sampled from the image of a "target" object we would like to find.*

We will use a distinctively coloured yellow object as our target. We will learn it's colour distribution (by fitting an Eigenmodel to it's pixel data) then examine pixels in a new image of the object to determine which pixels are likely to form part of that distribution form part of the target. This will be achieved by measuring "distance" from the Eigenmodel in standard deviations. This distance is the Mahalanobis distance covered in lectures.

First, load the target image and normalise it so the red, green and blue values range [0,1]instead of [0,255]. Although this normalisation is unnecessary we will do it for purpose of forming good habits. The image is a patch of colour cropped from a photo of the yellow object – it is the training data.

```
>> target=double(imread('~/cvprlab/testimages/target_yellow.bmp'))./255;
```

Create a matrix of RGB pixel values from the target image. Each column in the matrix will one pixel's RGB value, i.e. the matrix will contain three rows, for red green and blue respectively. It will contain a column for each of the 1457 pixels in this test image:

```
>> target_obs=[reshape(target(:,:,1),1,[])  ;  reshape(target(:,:,2),1,[])  ;
reshape(target(:,:,3),1,[]) ];
```

The variable target_obs now contains 1457 data points, from which we build an Eigenmodel:

```
>> target_e = Eigen_Build(target_obs);
```

We have now modelling the target's colour distribution, and will seek pixels of similar colour in a new "test" image to try to locate the pen. First, load the test image:

```
>> test = double(imread('kitchen.bmp'))./255;
```

Now obtain a matrix of RGB values; again, one column (data point) per pixel. Here there are 320x240 pixels, i.e. ~77k pixels:

```
>> test_obs=[reshape(test(:,:,1),1,[])  ;  reshape(test(:,:,2),1,[])  ;
reshape(test(:,:,3),1,[]) ];
```

We will now compute the Mahalanobis distance of each of the 77k data points from the Eigenmodel:

```
>> mdist =  Eigen_Mahalanobis(test_obs,target_e);
```

The resulting ‘mdist’ is a single row with 77k columns, each containing a value for each pixel that encodes distance from the colour model. We can reshape ‘mdist’ from a 1x76800 matrix back into a matrix of 240 x 320 in order to visualize the result.

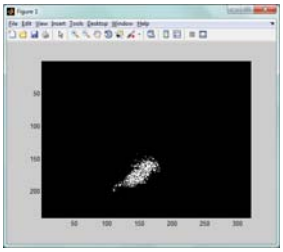
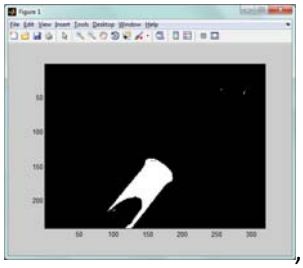
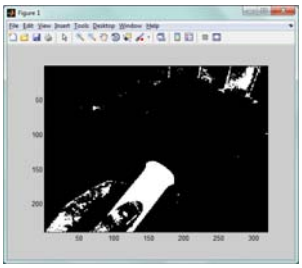
```
>> result = reshape(mdist,size(test,1),size(test,2));
```

The values in result will range from 0 to some large positive value. We can find the maximum value in ‘result’ and divide each pixel by it, to get a normalised result ‘nresult’ with pixels in range [0,1] which we can then display.

```
>> nresult=result./ max(max(result));
>> imgshow(nresult);
```

The darker pixels have lower Mahalanobis distance i.e. are closer to the colour model. The pen should therefore be picked out with darker intensity pixels. We can threshold the image at 3 standard deviations:

```
>> imgshow (result < 3);
```

3 std devs  , 30 std devs  , 60 std devs 

## Ex.4: Compute the Mahalanobis Distance manually *(10 minutes)*

We will now manually compute the Mahalanobis distance of one of the pixels from the Eigenmodel, as an exercise. First, select one of the pixels in the image. The first pixel in the 'test_obs' matrix will suffice:

```
>> x=test_obs(:,1)
x =
    0.2941
    0.2118
    0.0510
```

First, subtract the mean of the eigenmodel:

```
>> xsub = x-target_e.org;
```

Now use the equation discuss during lectures to compute the square of the Mahalanobis distance (i.e. the distance of the point 'x' from the eigenmodel in units of variance).

```
>> V= diag(target_e.val);
>> U= target_e.vct;
>> mdist_squared = xsub' * U * inv(V) * U' * xsub;
```

So the square root of 'mdist_squared' is the Mahalanobis distance of 'x' from the model:

```
>> sqrt(mdist_squared)
```

Which should correspond to the first column of the Mahalanobis distances previously computed in Ex.3:

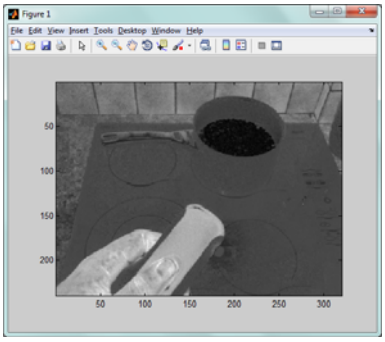
```
>> mdist(1)
```

## ***OPTIONAL*  Ex.5:  Repeat Ex.3 for the green object / saucepan of peas *(15 minutes)***

*Repeat Ex.3 to find the green region (saucepan of peas) in the kitchen image*

Using the alternative supplied target image ‘target_peas.bmp’

You should be able to visualise the Mahalanobis distance as follows:




which can be thresholded at 3 std deviations to get: