

# **Clab-2 Report**

ENGN6528

Han Zhang  
u7235649

01/05/2021

## Task-1: Harris Corner Detector

1. Read and understand the above corner detection code.
2. Complete the missing parts, rewrite them to “harris.py” as a python script, and design appropriate function signature.

The signature of the function *corner\_detect* is:

- the inputs:
  - *\_img*: the grayscale image to process,
  - *\_Ix2*: the square of the x derivative,
  - *\_Iy2*: the square of the y derivative,
  - *\_Ixy*: the product of the x derivative and the y derivative,
  - *\_k*: response function parameter,
  - *\_th*: the threshold,
- The output:
  - *locs*: an array with the same shape with *\_img* that corners are marked as 1 and the others 0.

The function calculates the degree of change according to the formula:

$$R = \det M - k(\text{trace } M)^2$$

The function also performs non-maximum suppression and thresholding, only the points with the locally highest degree of change above the threshold was selected.

3. Comment on the codes in block #5 and every line of your solution in block #7.

The codes in block #5 generates a Gaussian kernel as window function.

Please refer function *corner\_detect* for the comment of my solution in file *task1.py*,

*harris.py* or *Task1.ipynb*.

**4. Test this function on the provided four test images. Display your results by marking the detected corners on the input images.**

Harris-1.jpg: The original image is shown as Fig. 1 and the harris corner detected image as Fig. 2 below.

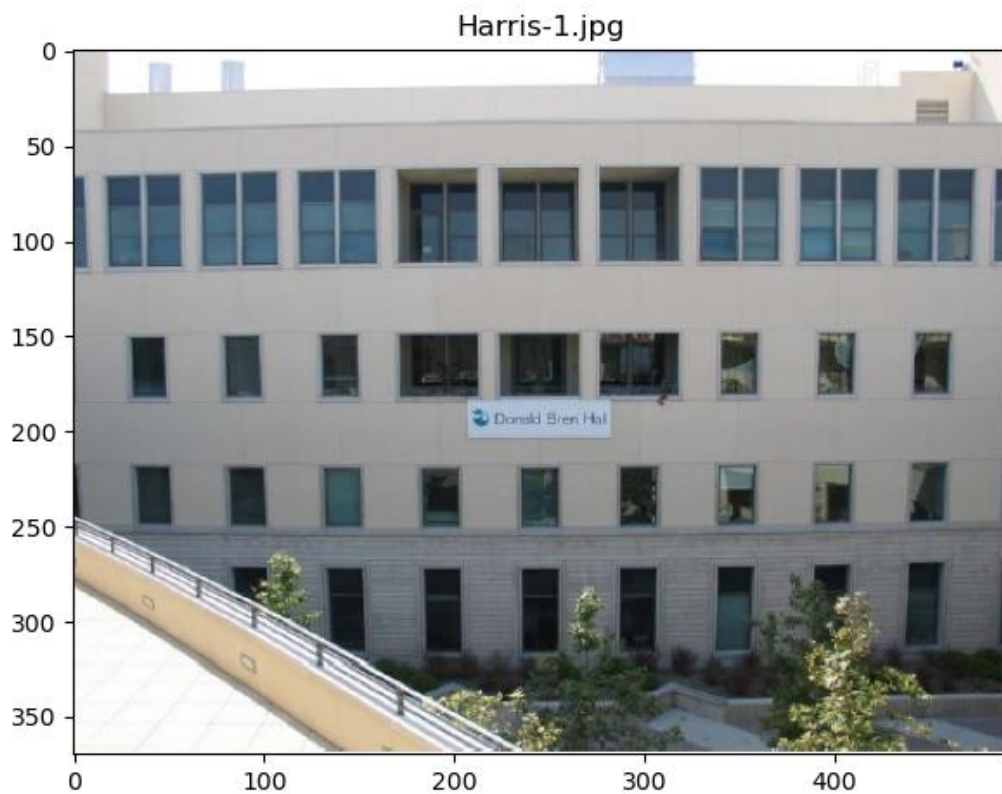


Fig. 1

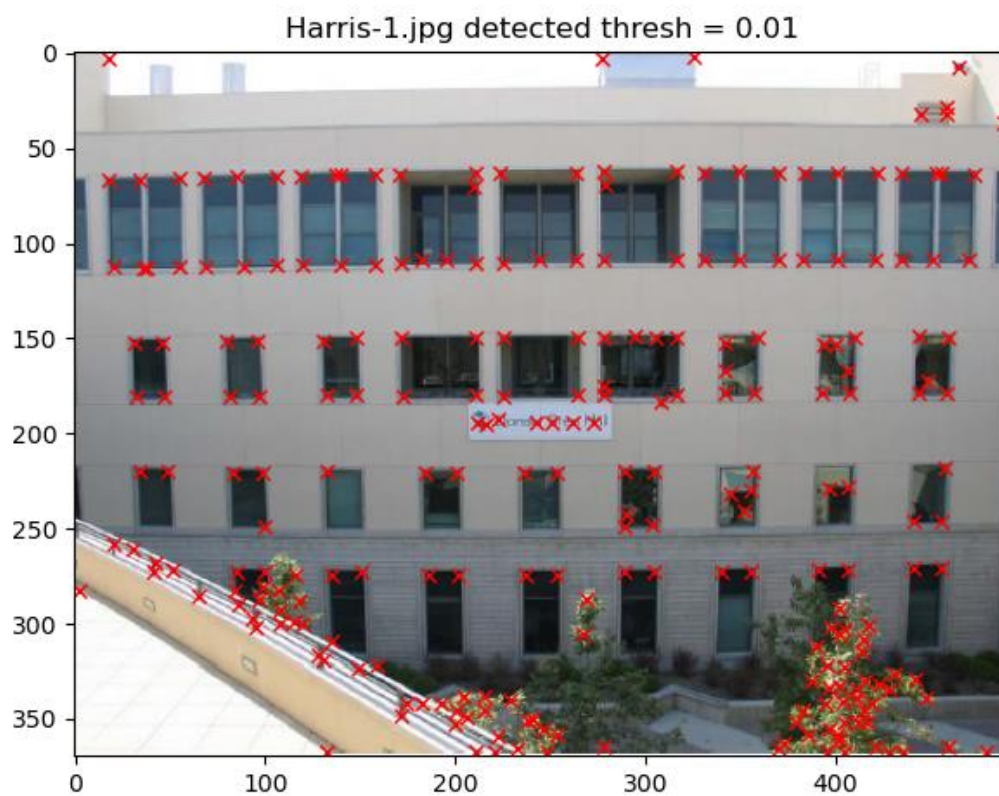


Fig. 2

Harris-2.jpg: The original image is shown as Fig. 3 and the harris corner detected image as Fig. 4 below.



Fig. 3

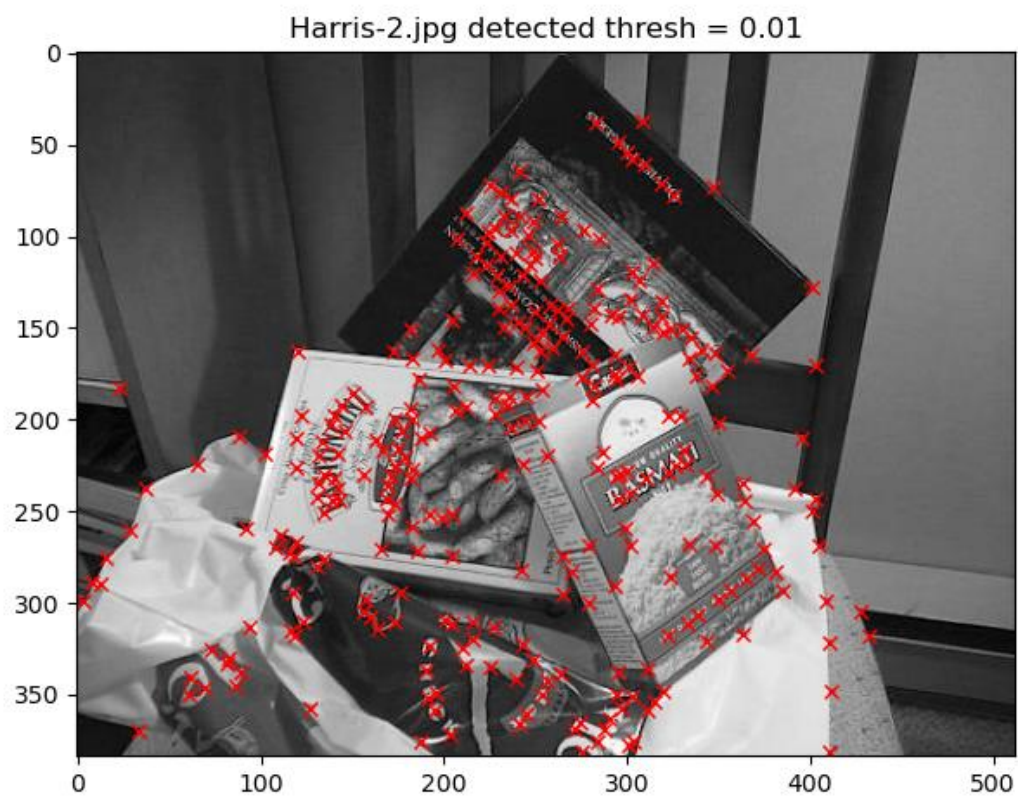


Fig. 4

Harris-3.jpg: The original image is shown as Fig. 5 and the harris corner detected image as Fig. 6 below.



Fig. 5



Fig. 6

Harris-4.jpg: The original image is shown as Fig. 7 and the harris corner detected image as Fig. 8 below.



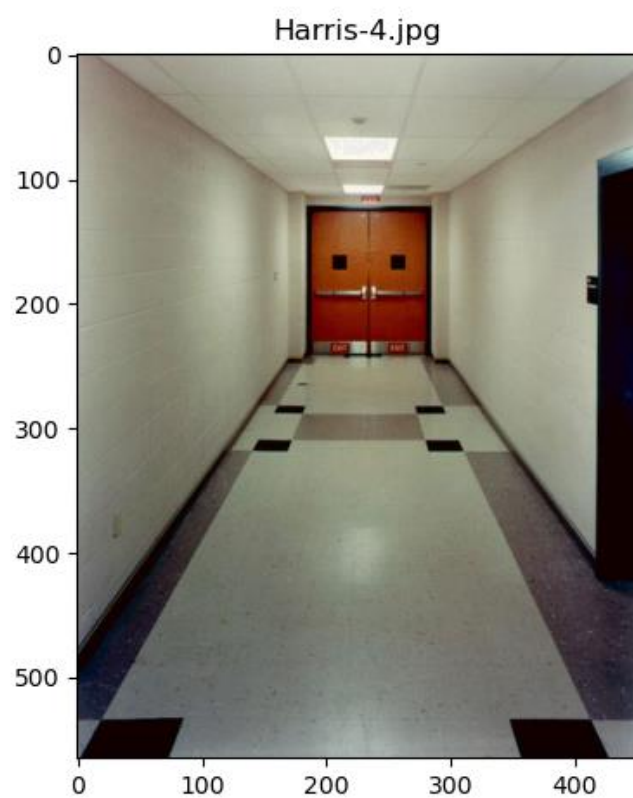


Fig. 7

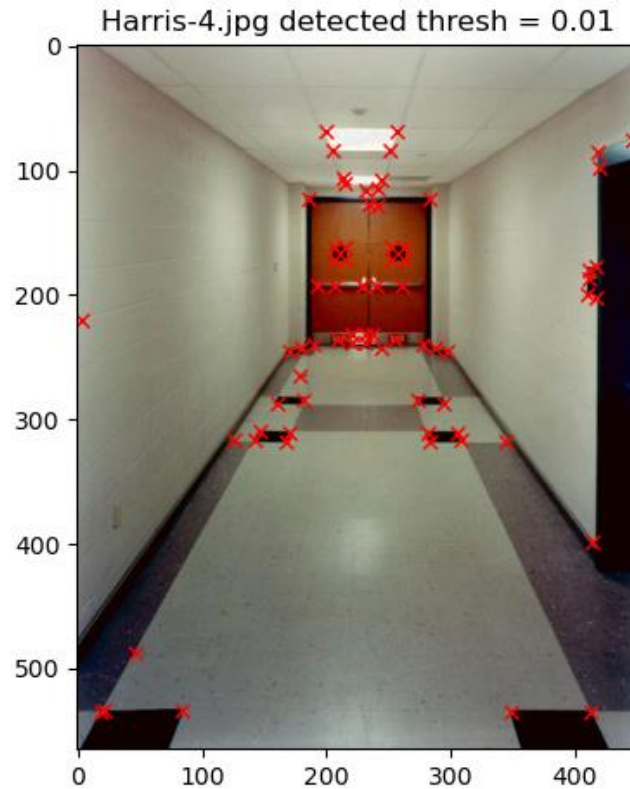


Fig. 8

**5. Compare your results with that from python's built-in function `cv2.cornerHarris()` and discuss the factors that affect the performance of Harris corner detection.**

The result of the inbuilt function is shown as Fig. 9 and the result of my function is shown as Fig. 10 below.

The factors that affect the performance of Harris corner detection are the threshold and the response function parameter  $k$ . Lowering the threshold allows more angles with lower contrast to be detected. Increase the value of the response function parameter  $k$  will reduce the sensitivity of corner detection, and reduce the number of corners detected.

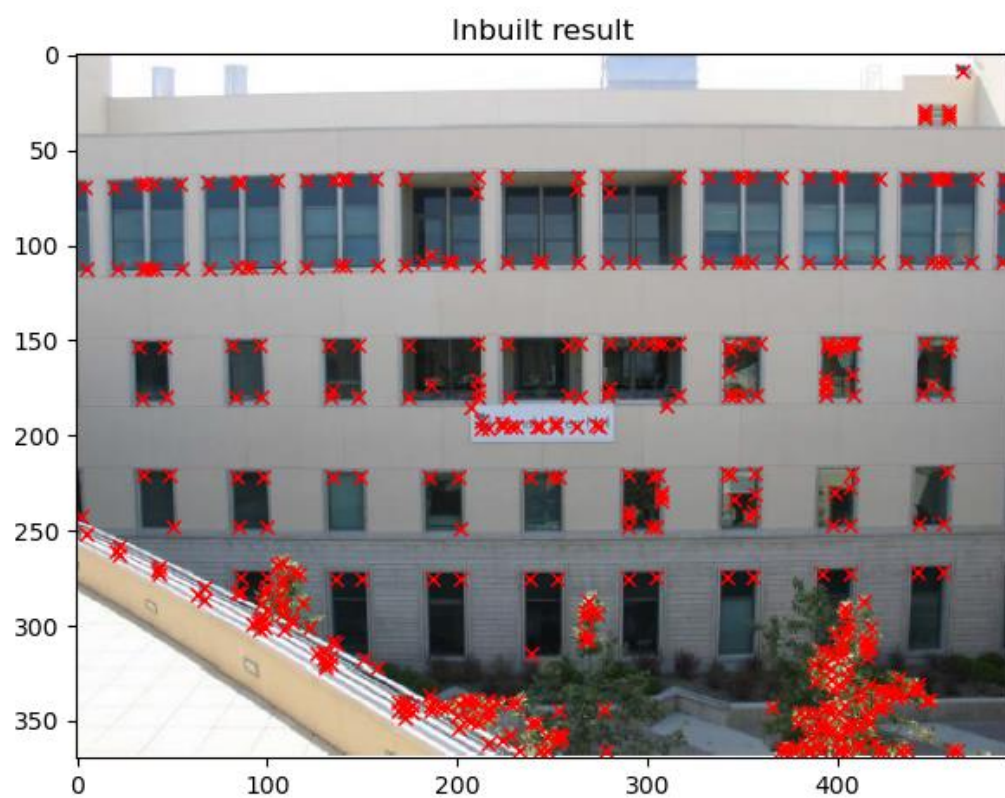


Fig. 9

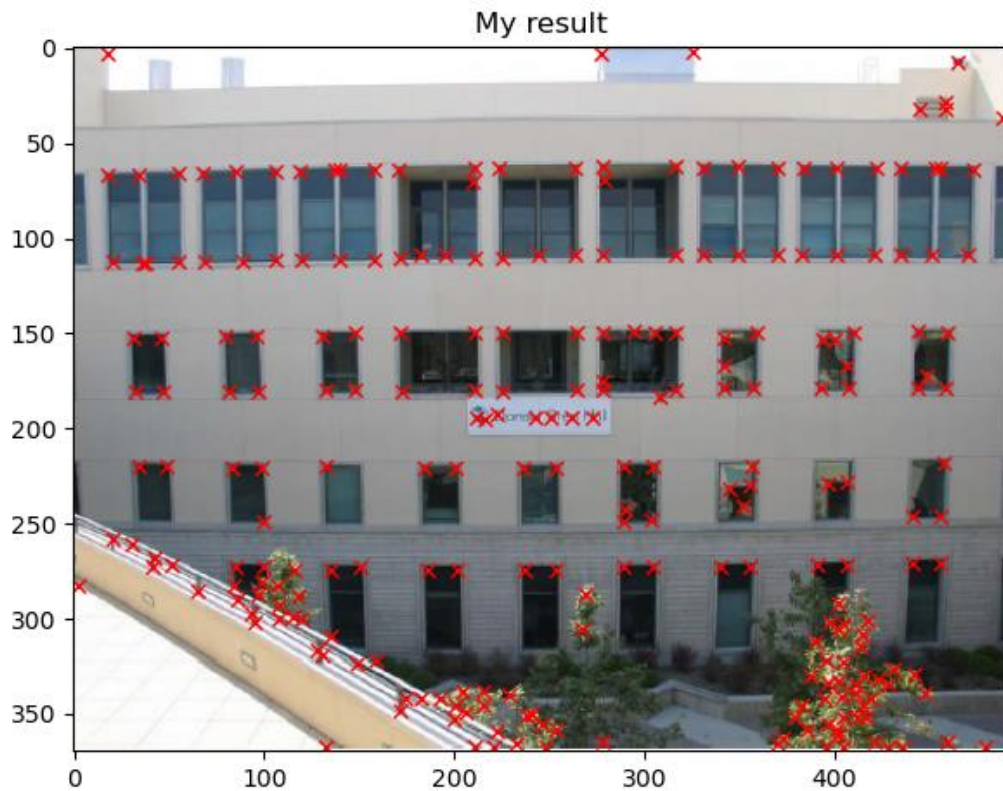


Fig. 10

**6. Test this function on 'Harris-5.jpg'. Analyse the results why we cannot get corners by discussing and visualising your corner response scores of the image.**

In this image there is only border but no corner. When the window is moving, only the degree of change in the horizontal direction is large, so it will be recognized as a line rather than a corner.

The result of the detected image is shown below as Fig. 11.

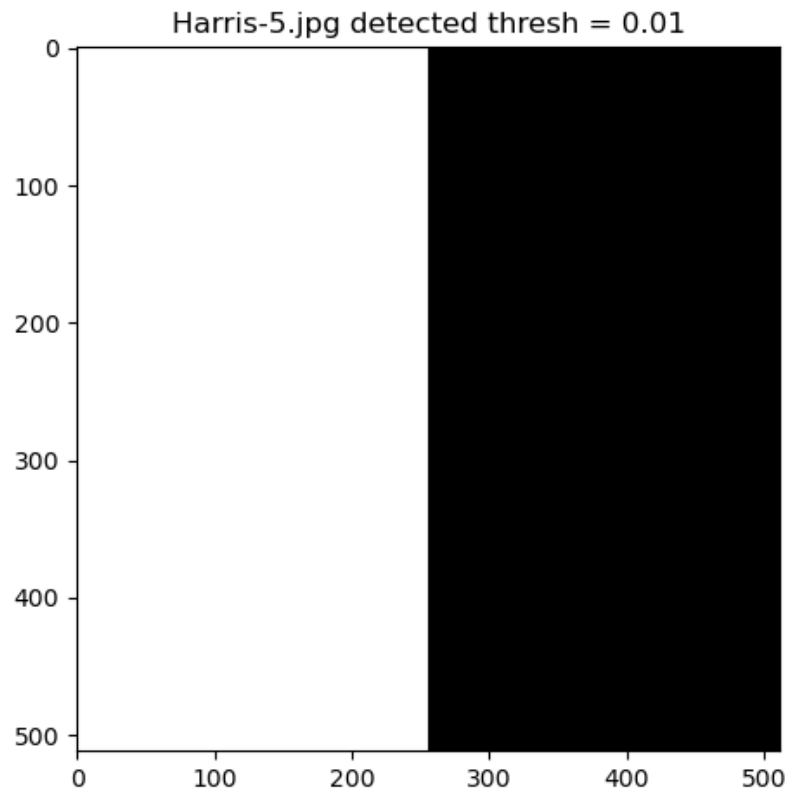


Fig. 11

- 7. Test this function on 'Harris-6.jpg'. Plot your harris corner detector results in your report and propose a solution to obtain the salient corners which is robust to noise.**

The detection result is shown below as Fig. 12. The image has a lot of noise which considered as corners by the detector.

One solution to solve this can be filter the image by a Gaussian filter to smooth the image, then apply harris corner detector. The result of the blurred image is shown below as Fig. 13. Only the central corner is exactly detected.

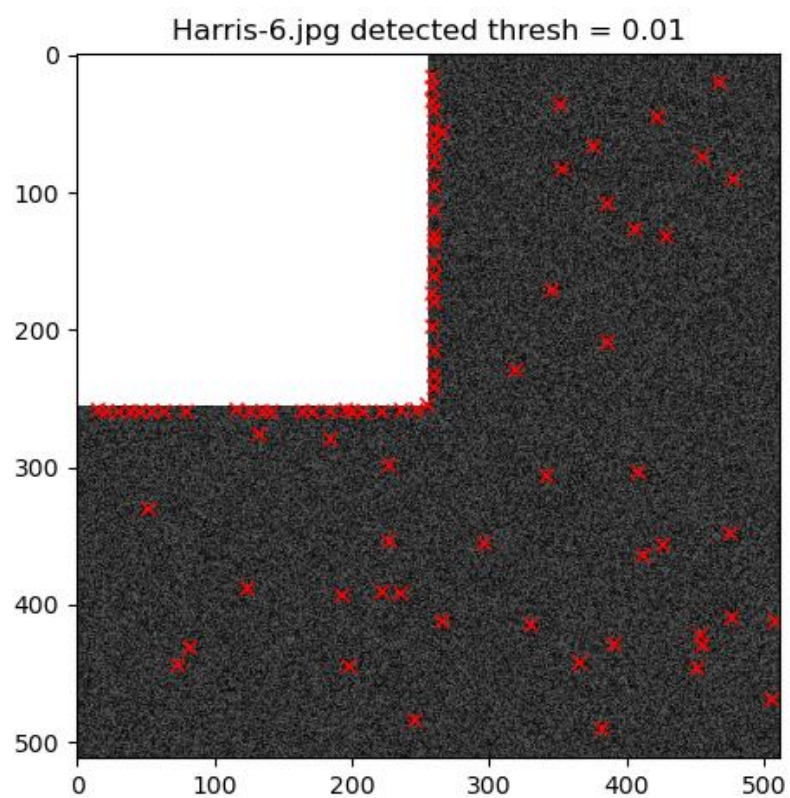


Fig. 12

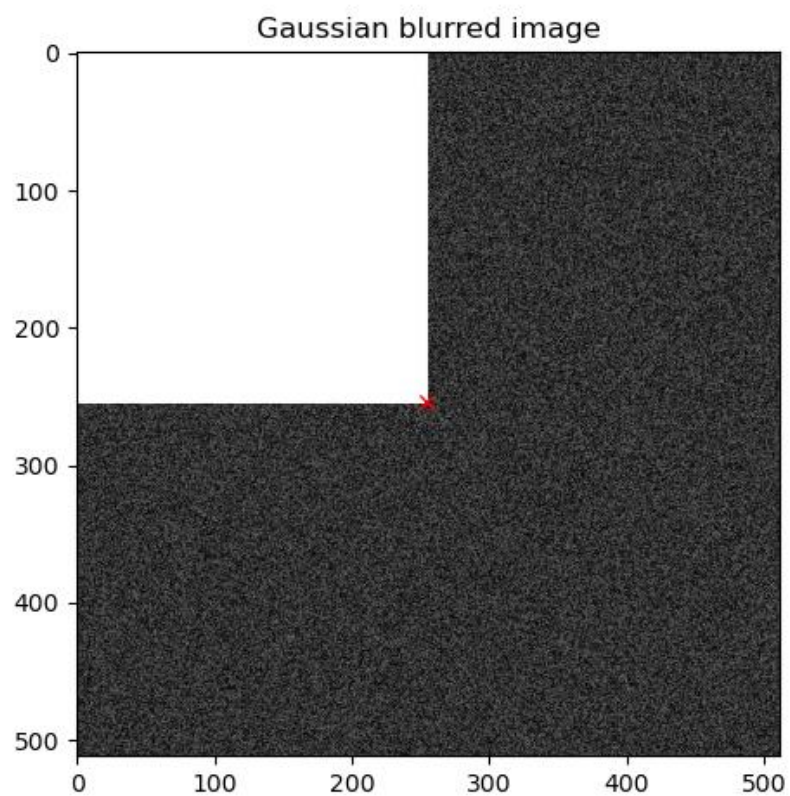


Fig. 13

## Task-2: K-Means Clustering and Color Image Segmentation

**1. Implement your own K-means function *my\_kmeans()*. The input is the data points to be processed and the number of clusters, and the output is several clusters of your data.**

The function *my\_kmeans()* receive the processed data and clusters number *k* as input and return a list that each element corresponds to the cluster of elements in the input data. The image should be processed by the function *get\_data()* whose parameters are the image to process and a Boolean to mark with coordinate or not.

The function first randomly generates *k* cluster centers. Each data element is assigned to the nearest cluster center by calculating the Euclidean distance between the element and the cluster centers, and then the cluster centers are updated by calculating the mean value of each cluster center.

**2. Apply your K-means function to color image segmentation. Each pixel should be represented as a 5-D vector.**

Process the image with function *get\_data()*. Set the parameter *withCoordinates* to true to include the coordinates, so the data is a 5-D vector. Set the parameter *withCoordinates* to false to exclude the coordinates, so the data is a 3-D vector. This may be different with the question, but just a different way to format the data. Then put the data to function *my\_kmeans()* as well as the cluster numbers *k* for the result.

The original image of peppers.png is shown as Fig. 14 below. The result of 4 clusters with coordinates is shown as Fig. 15, the result of 6 clusters with coordinates is shown as Fig. 16. The result of 4 clusters without coordinates is shown as Fig. 17 and the result of 6 clusters without coordinates is shown as Fig. 18.



The original image of mandm.png is shown as Fig. 19 below. The result of 4 clusters without coordinates is shown as Fig. 20, the result of 4 clusters with coordinates is shown as Fig. 20.

With or without coordinates will significantly affect the clustering results.

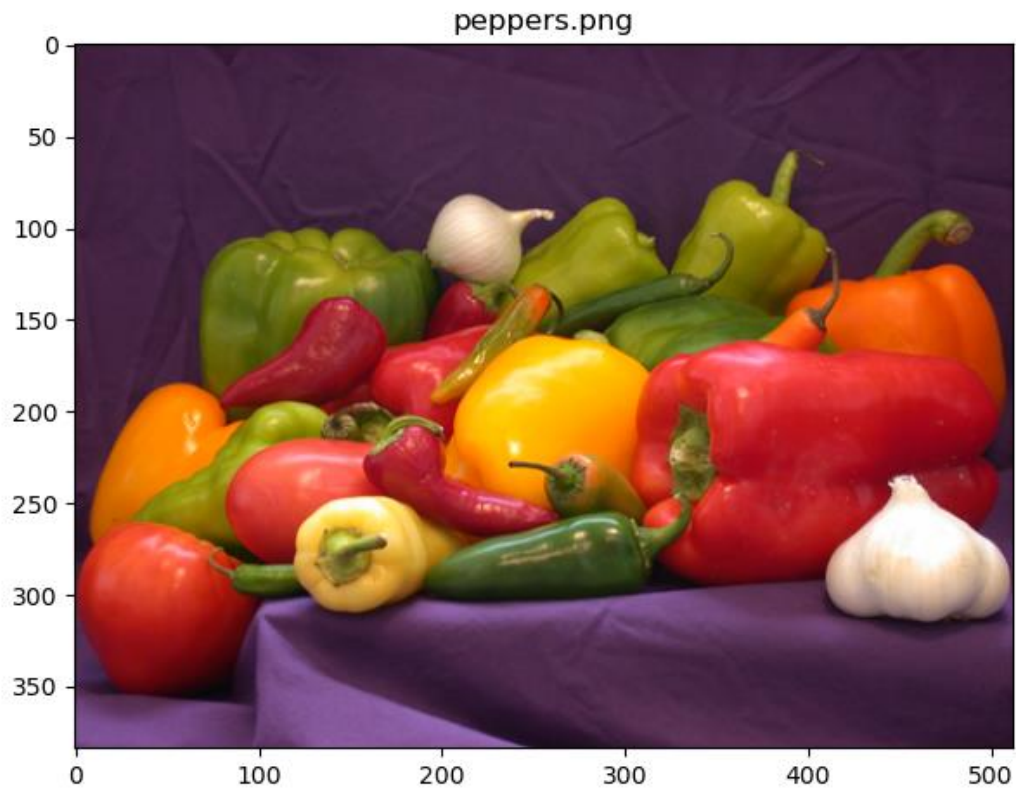


Fig. 14

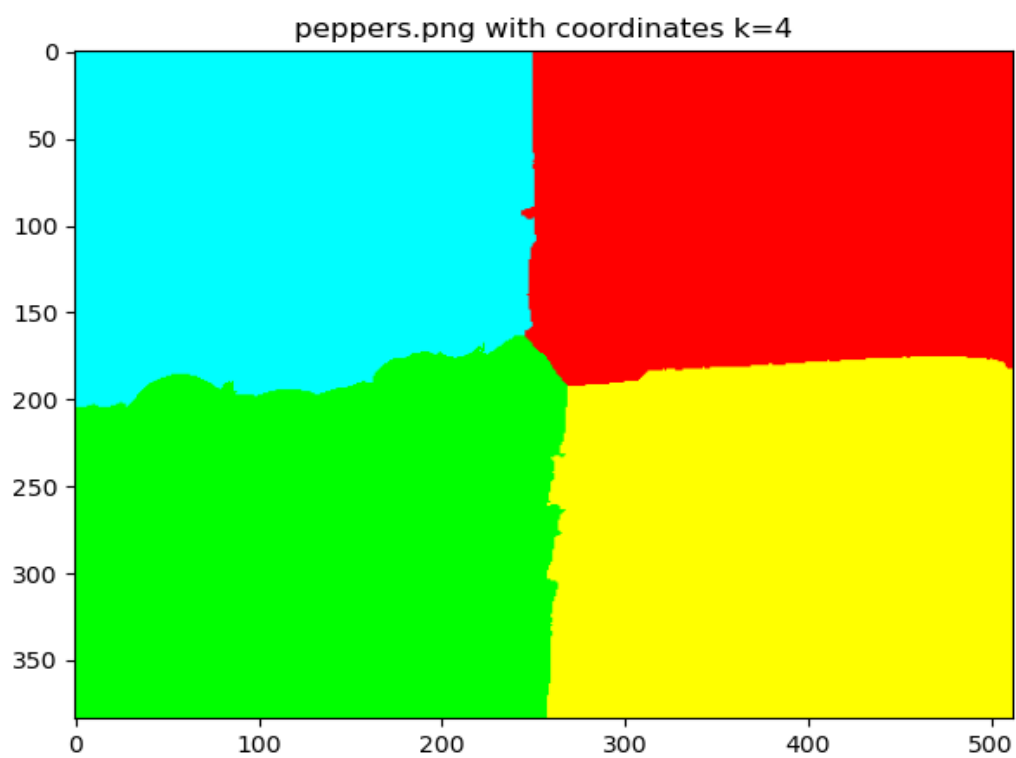


Fig. 15

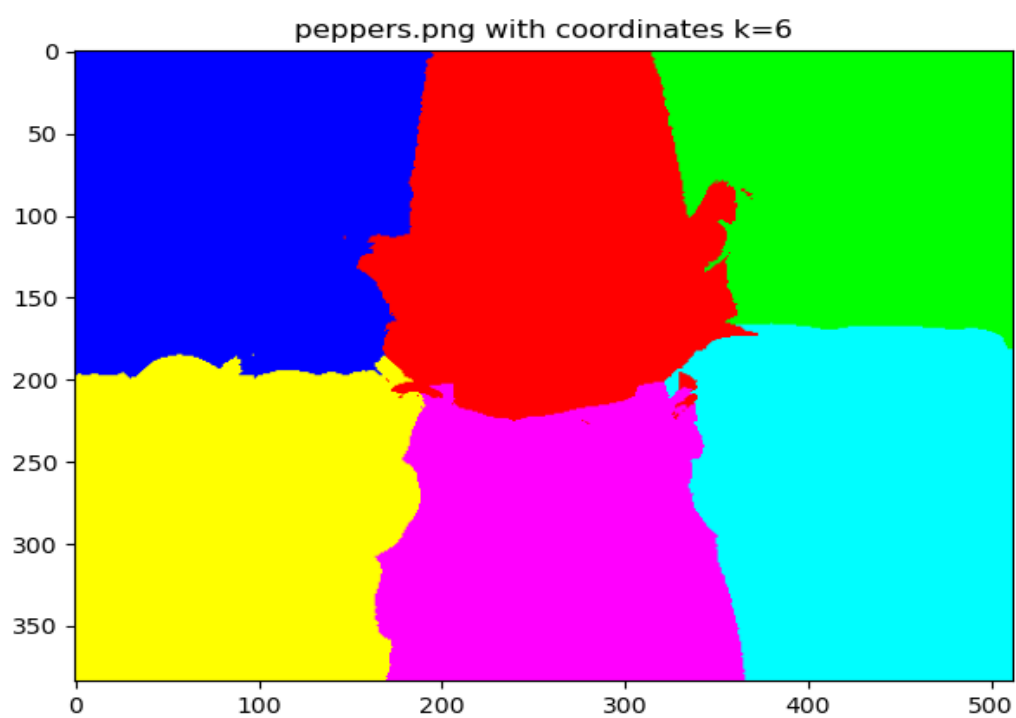


Fig. 16

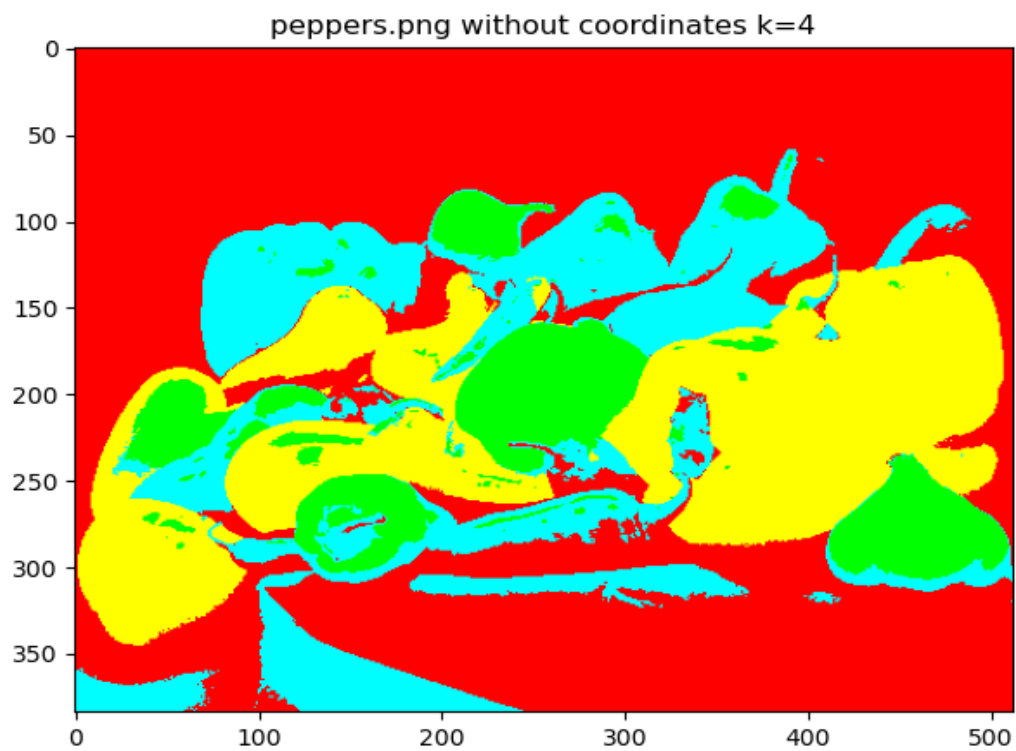


Fig. 17

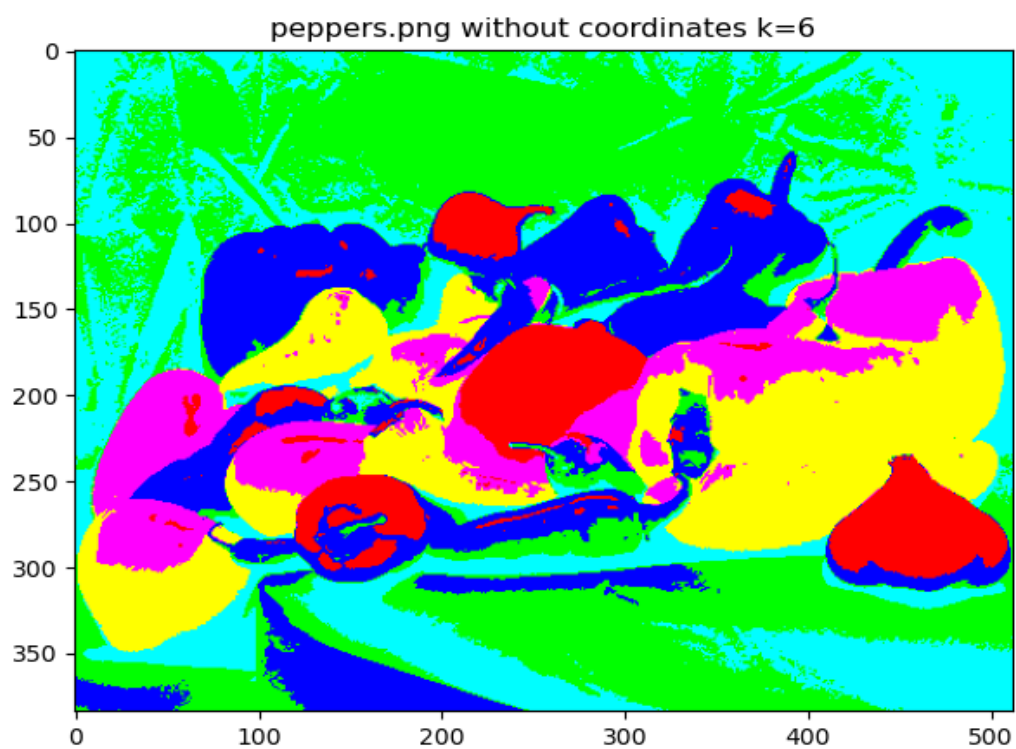


Fig. 18

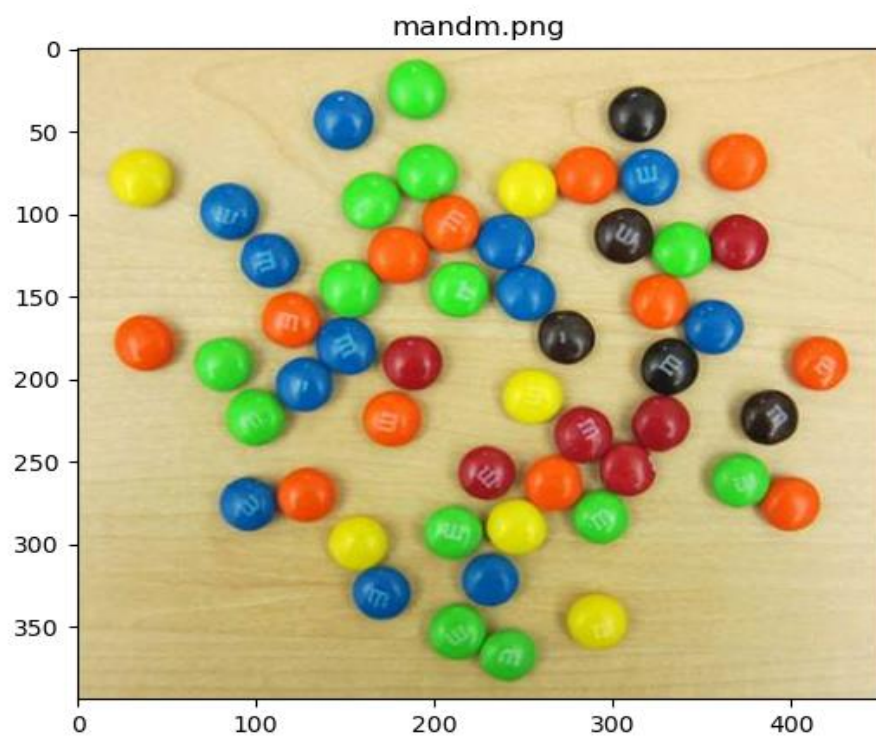


Fig. 19

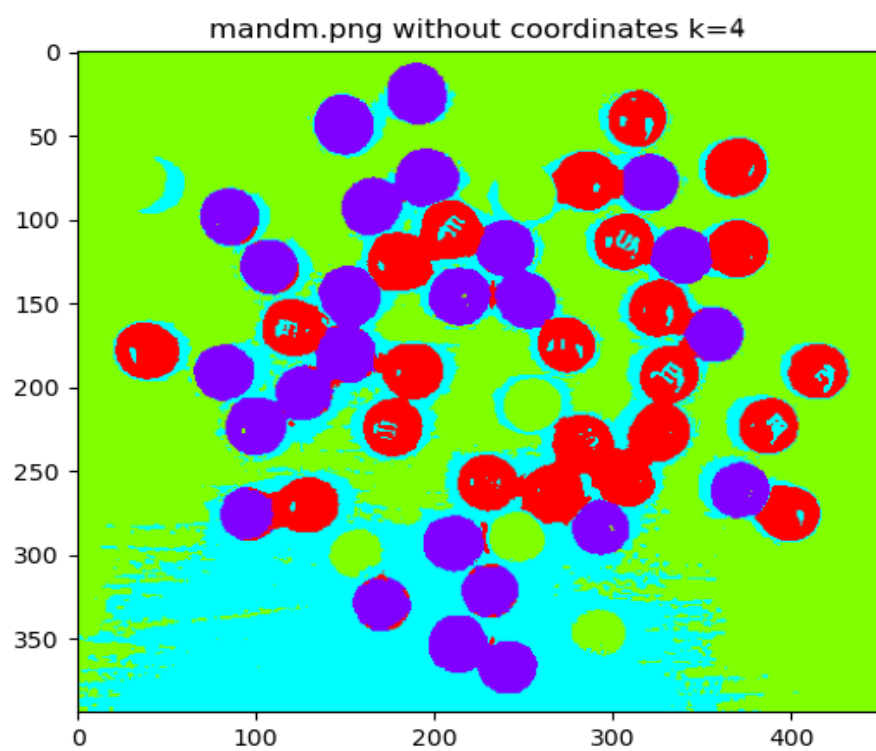


Fig. 20

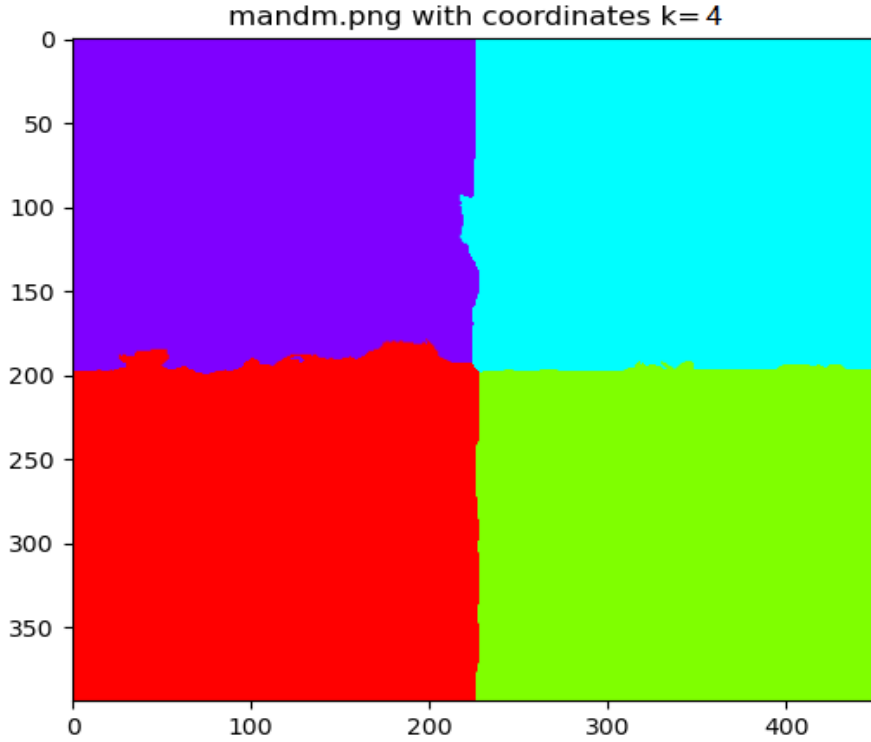


Fig. 21

**3. Please read the material, summarize the key steps in the report, and then implement K-means++ in your standard algorithm as a new initialization strategy. Compare the image segmentation performance of this new strategy, with that of standard K-means, using different numbers of clusters and the same 5-D point representation as that from previous question.**

For K-means++, the initialization of the cluster center will be more scattered. For  $k$  clusters, the key steps are:

1. Randomly pick one cluster center  $x$ .

2. Take a new center  $x$  with probability: 
$$\frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2}$$

3. Repeat step 2 until there are  $k$  cluster centers.

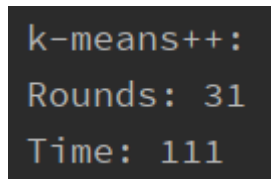
#### 4. Run the standard k-means algorithm.

For comparison I use the image mandm.png and set k to 3 and 5.

When  $k = 3$ , the performance for k-means++ is shown as Fig. 22, and the result shown as Fig. 23. The performance for standard k-means is shown as Fig. 24 and the result as Fig. 25.

When  $k = 5$  the performance for k-means++ is shown as Fig. 26 and the result as Fig. 27. The performance for standard k-means is shown as Fig. 28 and the result as Fig. 29.

It can be seen from the results that the performance of k-means ++ algorithm is better than the standard k-means algorithm. However, due to the randomness of the initialization of the clustering center, not every time the result is that k-means ++ algorithm is better than the standard k-means algorithm.



```
k-means++:  
Rounds: 31  
Time: 111
```

Fig. 22

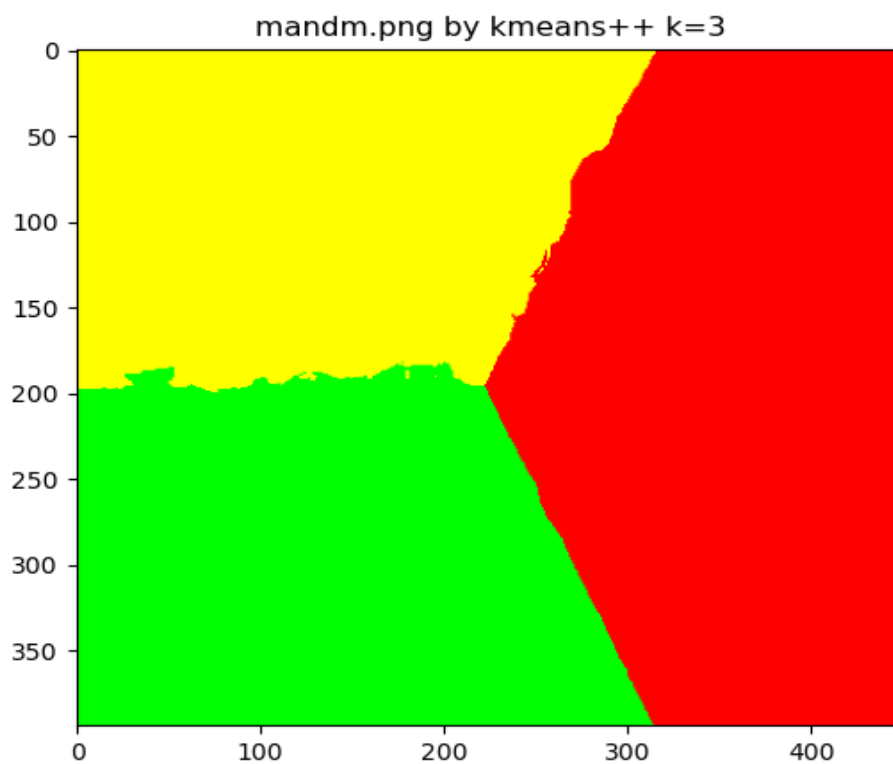


Fig. 23

```
std kmeans:  
Rounds: 32  
Time: 110
```

Fig. 24

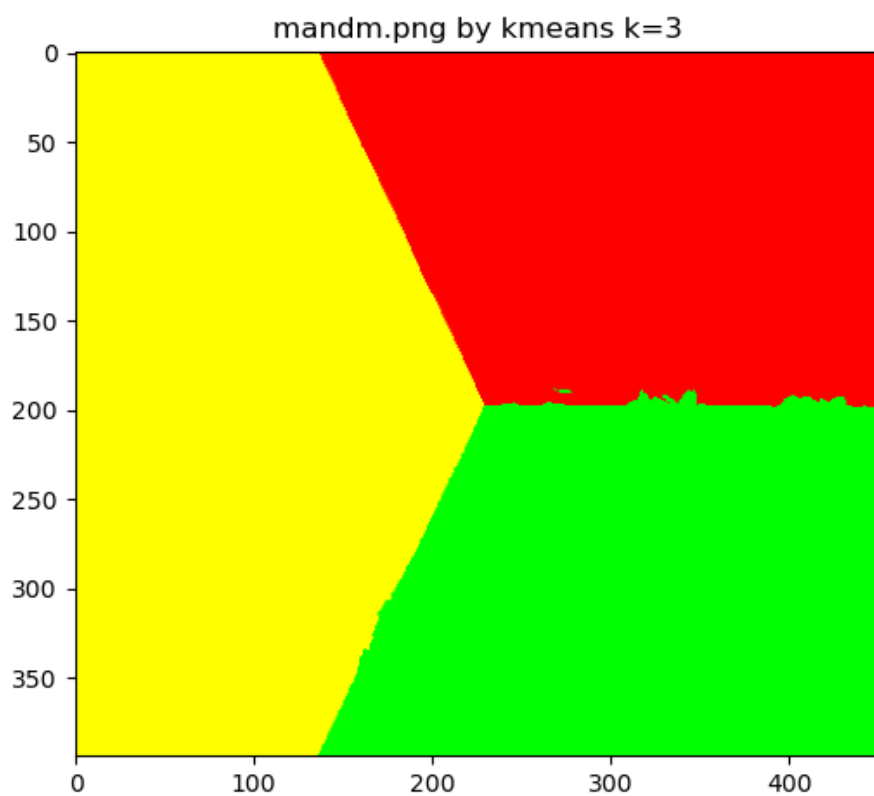


Fig. 25

```
k-means++:  
Rounds: 44  
Time: 256
```

Fig. 26



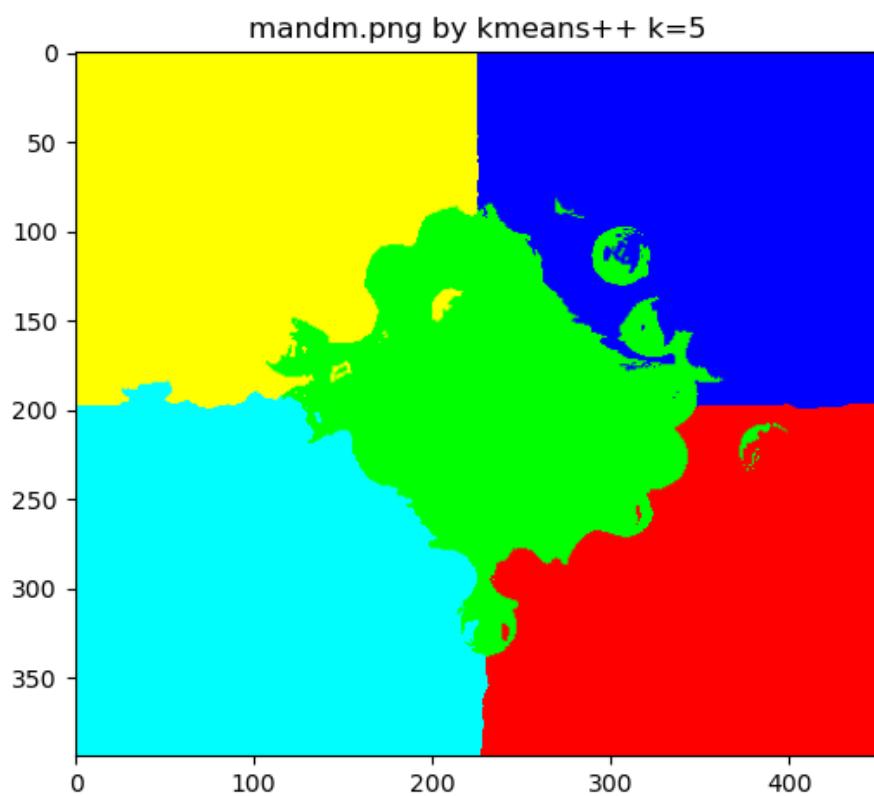


Fig. 27

```
std kmeans:  
Rounds: 58  
Time: 353
```

Fig. 28

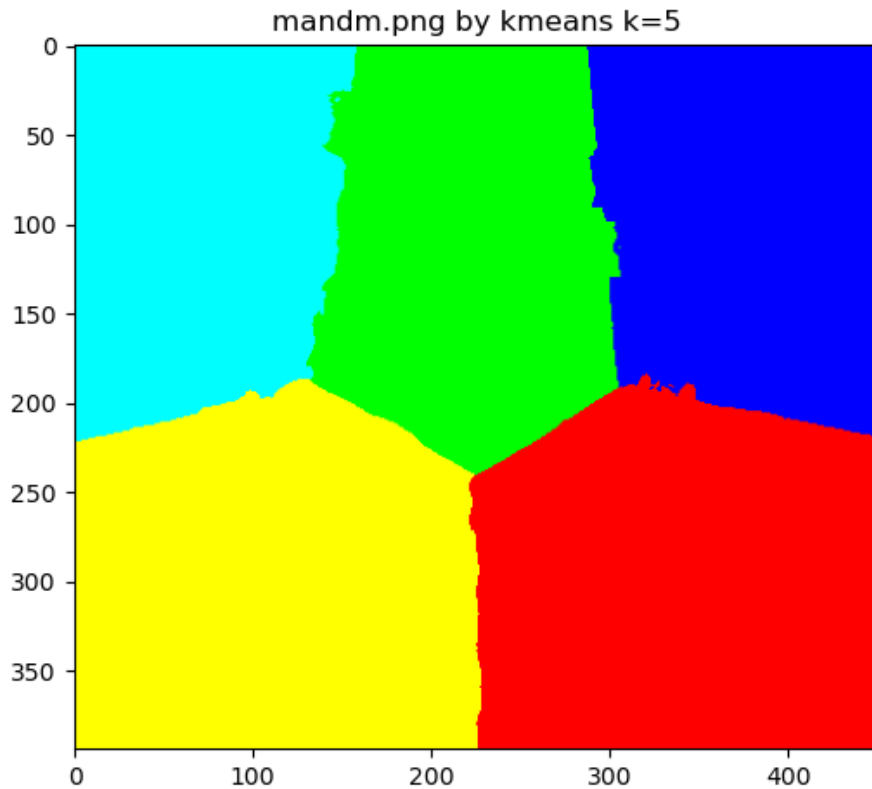


Fig. 29

### Task-3: Face Recognition using Eigenface

**1. Please unzip the face images and get an idea what they look like. Then please take 10 different frontal face images of yourself, convert them to grayscale, and align and resize them to match the images in Yale-Face. Explain why alignment is necessary for Eigen-face.**

Alignment can make the image form closer to each other and improve the accuracy of the generated Eigenvector, thus improving the accuracy. It can also make the operation and training easier and faster for large data sets.

**2. Train an Eigen-face recognition system. Specifically, at least your face recognition system should be able to complete the following tasks:**

**(1) Read all the 135 training images from Yale-Face, represent each image as a single data point in a high dimensional space and collect all the data points into a big data matrix.**

The test set already includes my photo. For the training data, the folder *trainingset* does not include my photo, the folder *trainingsetpp* includes 9 of my photo.

**(2) Perform PCA on the data matrix and display the mean face. Please read lecture notes and find a faster way to compute eigen values and vectors, explain the reason and implement it in your own code.**

The mean face is shown as Fig. 30 below.

The feature vector is the deviation of the training images and the mean face. Instead of directly performing eigen-decomposition on the whole covariance matrix, perform the process on each row so each row can get a weight vector since by calculating the projection of the unknown face on the feature vector, we can get the difference between the face to be detected and the mean face.

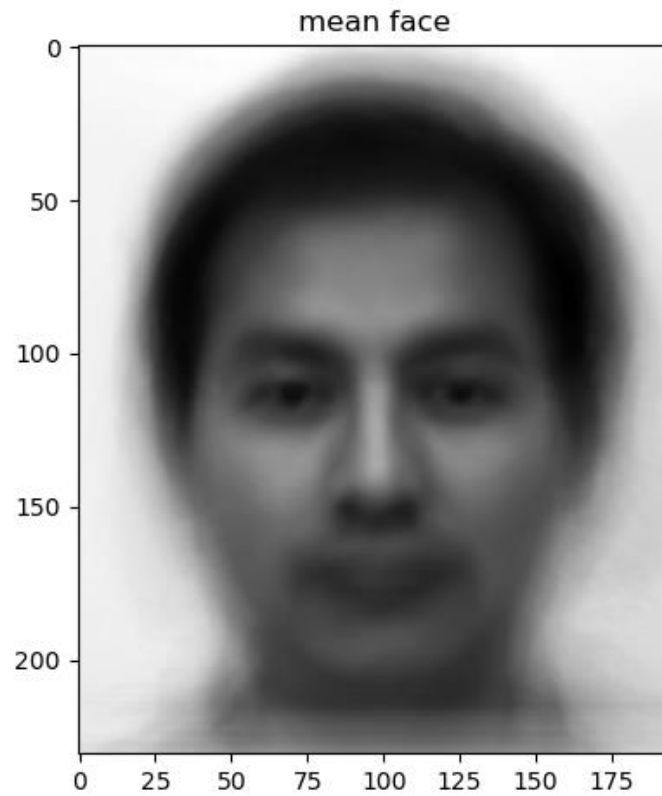


Fig. 30

**(3) Determine the top k principal components and visualize the top-k eigen-faces in your report.**

K is set to 15 so there are the top 15 eigenfaces shown as Fig. 31 below.

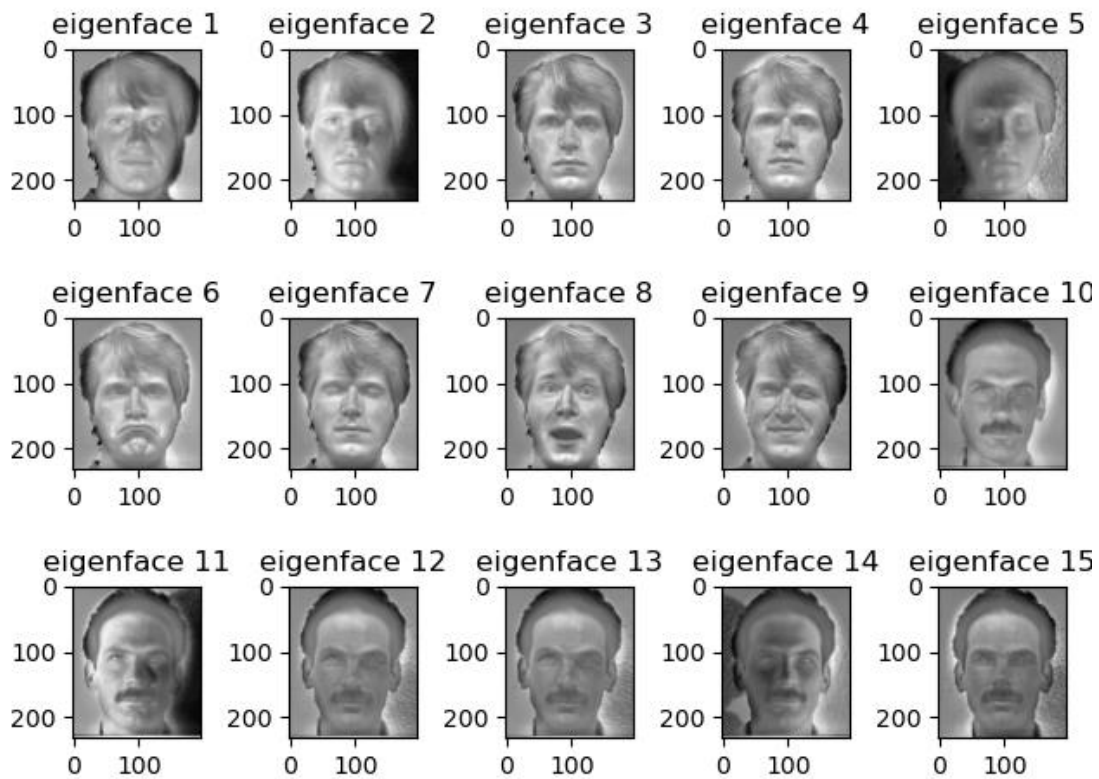


Fig. 31

**(4) For each of the 10 test images in the Yale-Face dataset, please read in an image as the reference one, and determine its projection onto the basis spanned by the top k eigenfaces. Use this projection as feature to perform a nearest-neighbor search over all 135 faces and find out the top three face images that are most similar to the reference one. Show these top 3 faces next to the test image in your report. Please report and analyze the recognition accuracy of your method.**

The training set for this does not include my photos. I loaded the first image in the testing set as reference, the recognition result is shown as Fig. 32 below.

The result of recognition is the same person, indicating the high accuracy of the method.

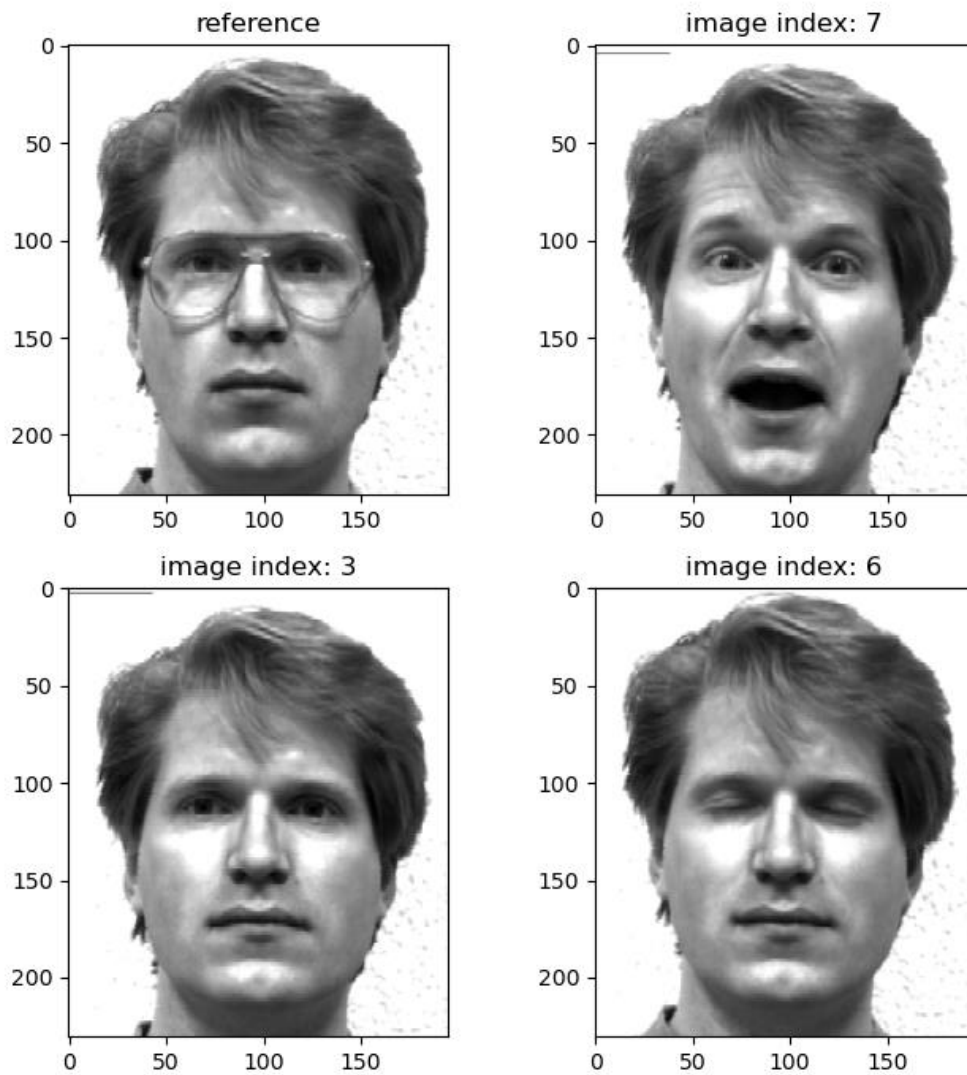


Fig.32

**(5) Read in one of your own frontal face images. Then run your face recognition system on this new image. Display the top 3 faces in the training folder that are most similar to your own face.**

The training set for this does not include my photos. I loaded my image in the testing set as reference, the recognition result is shown as Fig. 33 below.

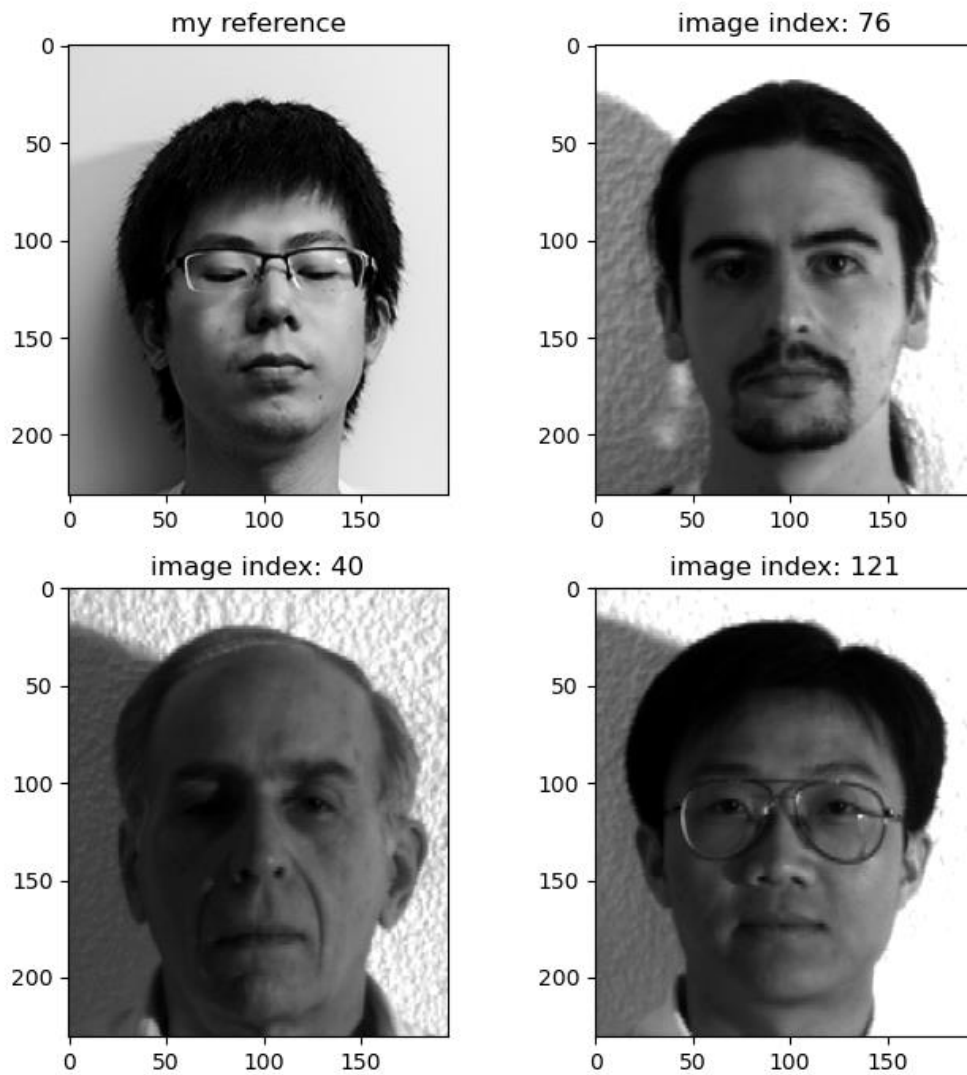


Fig. 33

**(6) Repeat the previous experiment by pre-adding the other 9 additional images of your face into the training set.**

The training set for this includes my photos. I loaded my image in the testing set as reference, the recognition result is shown as Fig. 34 below.

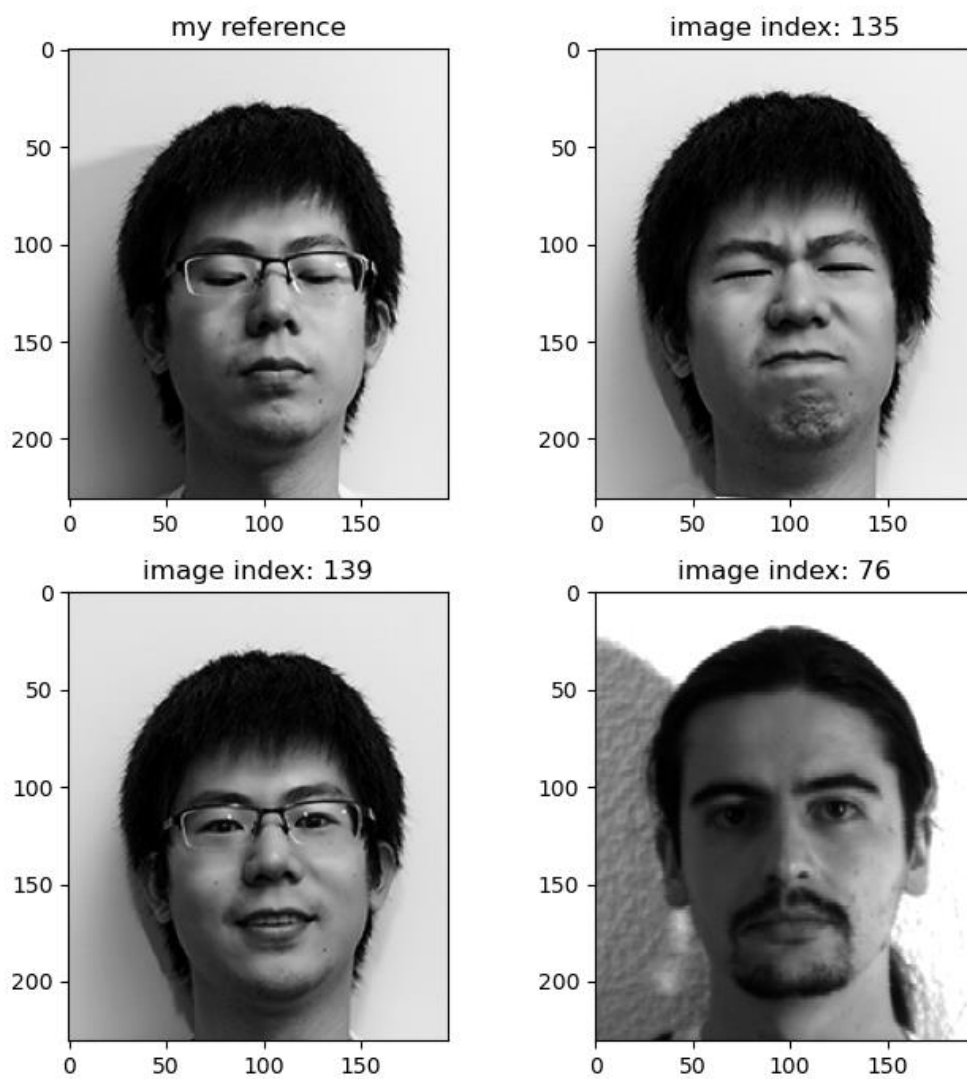


Fig. 34