# Task 1: Image Segmentation and detection

Sadanand Vithal Giraddi
School of Computer Science
University of Lincoln
Lincoln, United Kingdom
25895786@students.lincoln.ac.uk

***Abstract*— In this assessment, image processing techniques were implemented for object segmentation, and for each skin lesion image, the image processing techniques morphological operations are carried out for automatic segmentation of lesion object, and evaluation is done for the segmentation of each skin lesion image by calculating the Dice similarity score (DS). And with this, the three skin images ((ISIC_0000019, ISIC_0000095, and ISIC_0000214) and final segmented lesion binary images were shown, and a bar graph is plotted for all the 60-skin final segmented images and also evaluated the mean and standard deviation of dice scores of all the 60 images.**

## I. Introduction

The skin lesion dataset zip folder is provided, which contains the two folders: 1) org data folder and 2) GT folder. There is a total of 120 images among those the 60 are skin lesion color images and the other 60 are the corresponding binary masks of ground-truth segmentation [2].

**Image segmentation:** The process of segmenting a digital picture into various regions/ segments that represent objects or relevant sections of things is known as segmentation [2]. In other words, The process of segmenting an image into portions or regions is known as image segmentation [1]. The features of the pixels in the picture are frequently used to divide the image into portions. The two common approaches for image segmentation are edge-based and region-based. The edge-based approaches always look for edges in the picture (Prewitt, Sobel, Laplacian, Canny, and advanced local boundary cues are the image derivatives) and for obtaining the borders between the regions the edges can be linked. And the region-based approaches are the techniques in which it determines the regions directly and they are thresholding, adaptive threshold, watershed, and otsu.

**Morphological operations:** Morphology refers to a group of image processing methods that work with pictures depending on their forms. Each pixel in the image is changed based on the value of other pixels in its vicinity in a morphological procedure. You may create a morphological operation that is sensitive to certain forms in the input picture by adjusting the size and shape of the neighborhood [1].

**Dice similarity score:** It is defined as the twice of shared information (i.e. the intersection) which is divided by the sum of the cardinalities.
For two sets of M and N, the dice similarity score is : dice(MN) = $2 * | M \cap N | \ / | M | + | N |$
The dice score calculated will be between 0 and 1. If dice score is 1, then segmentation matches perfectly with the ground truth of the mask. And If the dice score is 0, there is no overlap between segmentation and ground truth of the mask [2].

## II. Implementation of Steps

1) Reading the skin lesion dataset and unzip the folder to see the org data containing the 60 skin lesion color images and other GT data containing the corresponding binary masks of ground truth segmentation.
2) Giving the path to reading the folder and using the 'for loop' functions for evaluating the 60 images of skin lesion color images and performing the morphological operations for segmenting the images.
3) The skin lesion images are segmented based on region-based techniques by using otsu's method and segmenting the images using logical operation.
4) The skin color images are converted from RGB images to gray images because of reducing the complexity of the 3D pixel values of the RGB images to 1D values(one-dimensional) for segmenting purposes and thresholding is a simpler method.
5) The gray images are then converted to the binary images and then binary images We choose a threshold value, and then all grey levels below that value are categorized as 0 (black, i.e. background), whereas grey levels equal to or greater than the threshold value are classed as 1. (white i.e foreground).
6) Now binary images are made to complement. Each color channel in the produced image complements the original image's corresponding color channel. Previously dark areas, such as dirt, become lighter.
7) And the evaluation is done by calculating the dice score, mean value, and standard deviation. And plotted the bar graph for the segmented object lesion images.
8) The dice similarity scores are than compared with the segmented object lesion images with the ground truth segmentation provided in the dataset and calculated the mean and standard deviation of dice scores for all the 60 images.

## III. Report Results

1) For the three skin images ((ISIC_0000019, ISIC_0000095, and ISIC_0000214) and final segmented lesion binary images are:

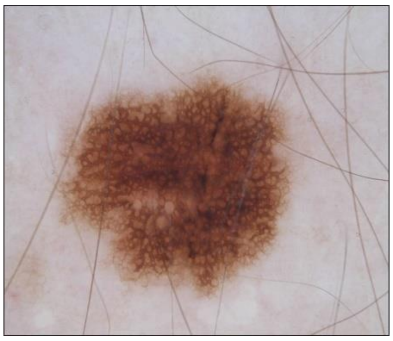


Fig.1: skin lesion image (ISIC_0000019)

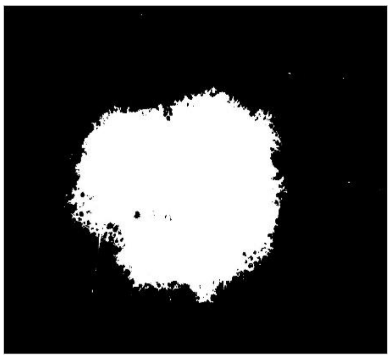


Figure 1.1: Segmented object lesion image (ISIC_0000019).
Dice score for Segmented object lesion image is (ISIC_0000019) = 0.8335

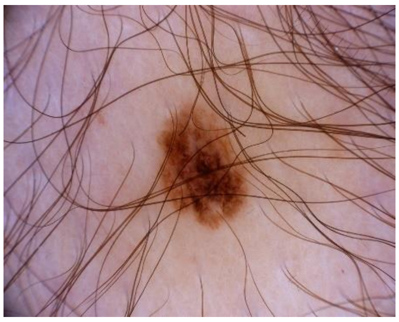


Fig.2: skin lesion image (ISIC_0000095)

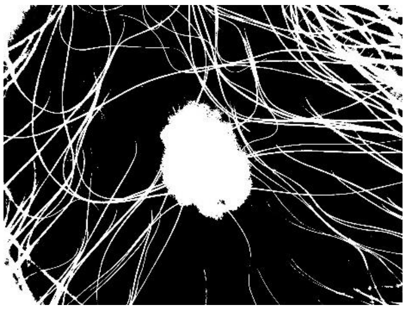


Figure 2.1: Segmented object lesion image (ISIC_0000095).
Dice score for Segmented object lesion image is (ISIC_0000095) = 0.4034

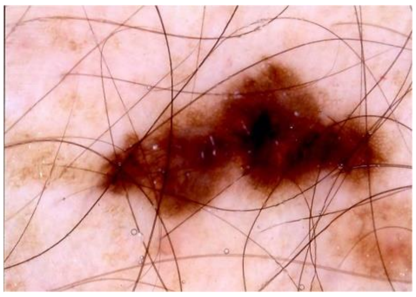


Fig.3: skin lesion image (ISIC_00000214)

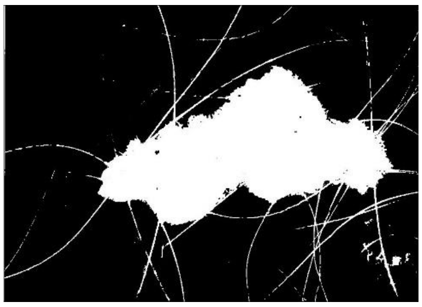


Figure 3.1: Segmented object lesion image (ISIC_00000214).
Dice score for Segmented object lesion image is (ISIC_0000019) = 0.8554

2) <u>Bar graph</u>: bar graph is provided for all the 60 segmented object lesion images in which x-axis represents the number of the images and y-axis represents the corresponding dice scores.

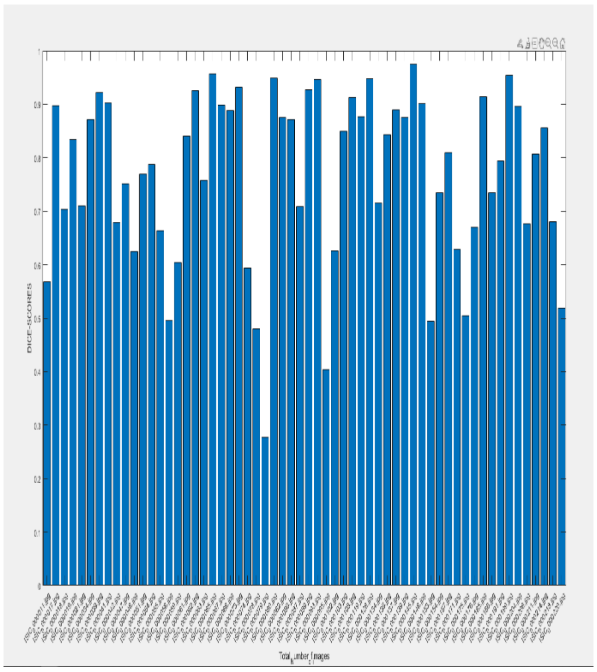


Figure 4: bar graph plotted for all the dice score values.

**SADANAND VITHAL GIRADDI**
**ID – 25895786**
**Computer Vision Assignment**

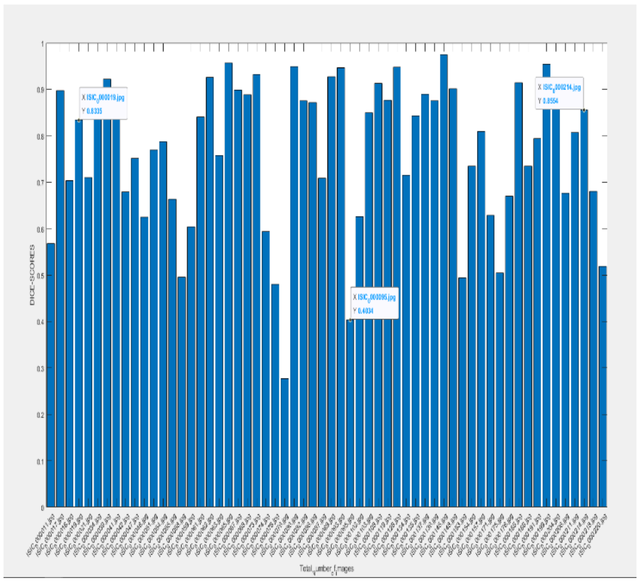
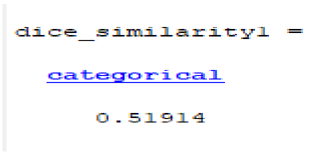


Figure 5: bar graph plotted for the three skin images ((ISIC_0000019, ISIC_0000095, and ISIC_0000214) of its respective dice score values is shown in the plot.

3)

**Mean value: 0.7682**
**Standard deviation value : 0.1603**
**Overall dice similarity score :0.51914**

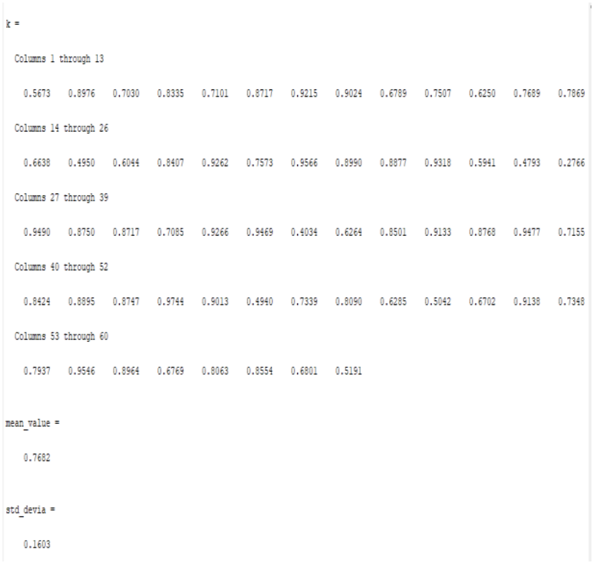


```
dice_similarity1 =

  categorical

      0.51914
```

Figure 6: overall dice similarity score.



```
k =

  Columns 1 through 13

    0.5673    0.8976    0.7030    0.8335    0.7101    0.8717    0.9215    0.9024    0.6789    0.7507    0.6250    0.7689    0.7869

  Columns 14 through 26

    0.6638    0.4950    0.6044    0.8407    0.9262    0.7573    0.9566    0.8990    0.8877    0.9318    0.5941    0.4793    0.2766

  Columns 27 through 39

    0.9490    0.8750    0.8717    0.7085    0.9266    0.9469    0.4034    0.6264    0.8501    0.9133    0.8768    0.9477    0.7155

  Columns 40 through 52

    0.8424    0.8895    0.8747    0.9744    0.9013    0.4940    0.7339    0.8090    0.6285    0.5042    0.6702    0.9138    0.7348

  Columns 53 through 60

    0.7937    0.9546    0.8964    0.6769    0.8063    0.8554    0.6801    0.5191


mean_value =

    0.7682


std_devia =

    0.1603
```

Figure 7: dice similarity score of all the segmented object lesion images with mean value and standard deviation.

## REFERENCES

[1]. Gonzalez, R.C. and Woods, R.E., Digital Image Processing, 2nd edition (Prentice Hall, 2002), Ch 10
[2]. Image segmentation lecture, University of Lincoln blackboard, computer vision week 2 lecture. [Prof. Xujiong Ye]

# Task 2: Feature calculation

*Abstract*— **In this assessment, the ImgPIA.jpeg image is provided in which it deals with feature extraction of the image area in both the spatial and frequency domains. And the features were selected for both the radius and direction. And the spectral approach methods are used for the feature extraction they are: energy, correlation, contrast, homogeneity, Angular second moment and inverse difference moment. From the image histograms of the co-occurrence matrix (provided by Haralick paper). And selecting the five features from the gray level run length matrix.**

## INTRODUCTION

**Texture Analysis:** It is the property which represent the surface and structure of an object [1].It consist of texture primitives and are called as texels. And the texture description is based on the tone which describes the pixel intensity and the structure describes about the spatial relationships between primitives. The spectral approach method is one way of analysing the texture.

**Spectral Approach method:** In the frequency domain, texture may be examined. If textures are periodic patterns, use periodic functions to examine them. The frequency domain as a whole comprises much data as the image itself [1].
The frequency-domain coordinates in polar coordinates expressed as:

$$S(r)=\sum_{\theta=0}^{\pi}S(r,\theta)$$

$$S(\theta)=\sum_{r=0}^{N/2}S(r,\theta)$$

Where S(r) gives the high and low frequencies at each angles. S(0) gives the frequency distribution content in specified direction. And two approaches are used they are Angular second moment and the inverse difference moment are considered as the features of spectral approach methods are used to the values of gray co-occurrence matrix.

**Co-Occurrence Matrix**: Under the premise that the spatial relationship between the grey tones of a picture f contains the textural information [1].
The textural information is represented by a matrix of relative frequencies P(a,b) in which two adjacent components separated by distance d, one with grey tone a and the other with grey tone b, appear in the image [1].

**Gray level run lenth(GLRL):** set of the image cells having the same gray tone is referred as gray level run and in the gray level run length the total number of image cells of the elements belongs to the same gray level run [1].

GLRL Features: short run emphasis, grey level non-uniformity, long runs emphasis, run percentage and run lengths non-uniformity [1].

## IMPLEMENTATION STEPS

**TASK 2.1**

1) Reading the ImgPiA.jpeg

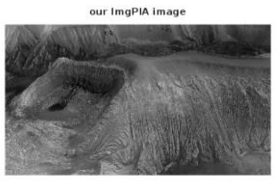


Figure 1: reading the given image

2) Transforming the image using Fourier transform function of the MatLab command.

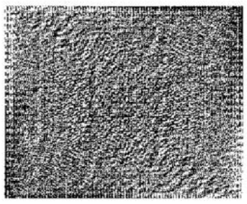


Figure 2: Transform of an image using fft matlab command.

3) Segmenting the image into four different section and transforming the image for four different section. for applying fourier transform function and the inverse fourier transform function.

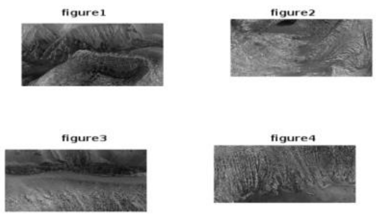


Figure 3: Segmenting the image into four different section

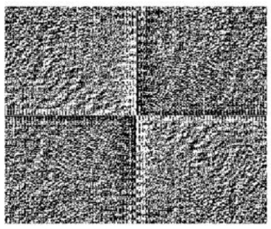


Figure 4: transformation of an image into four different sections.

4)Now applying the discrete fourier transform function for four different section and measuring the minimum and maximum values of transform amplitudes.

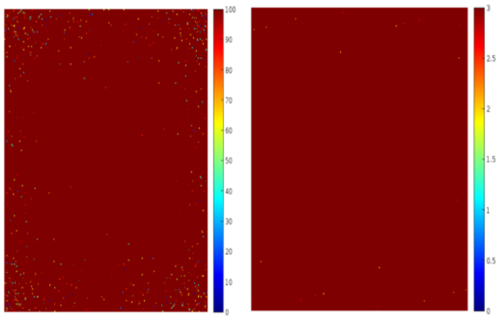


Figure 5:  looking at the phases of the colored map.

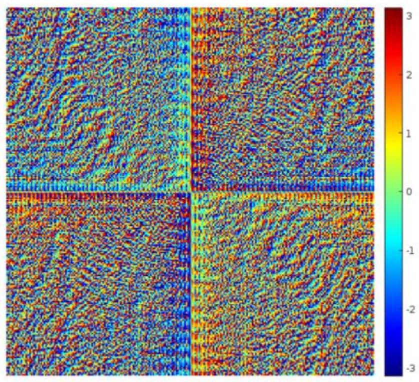


Figure 6: the obtained image by using the discrete fourier transformation of an image into four different sections.

5)    Now applying the inverse-fourier transform function for four different section

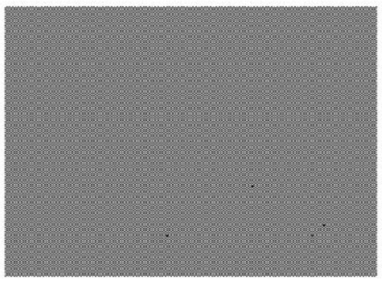


Figure 7: inverse transform function applied for the image.

6) Now selecting the feature image for by giving different offset values and testing it at different radius angles. The offset values that I have chosen are offset values= 2,4,6,8 and tested at four different angles they are  0 degrees, 45 degrees, 90 degrees and 13degreesee. And also learned the feature images of changing at each different offset change with different angles.

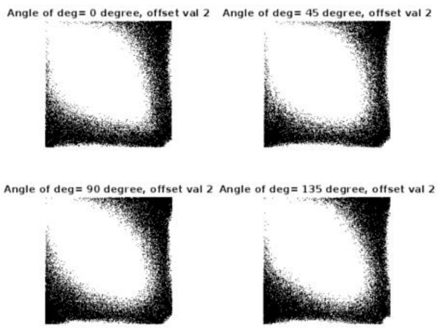


Figure 8: co-ocurrence matrix for offset value 2 and four different degrees.

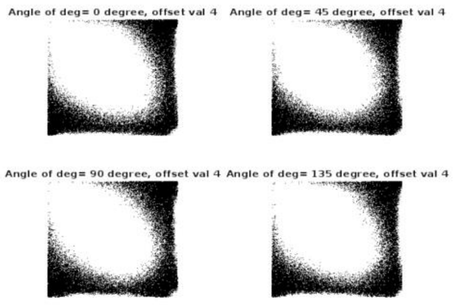


Figure 9: co-ocurrence matrix for offset value 4 and four different degrees.

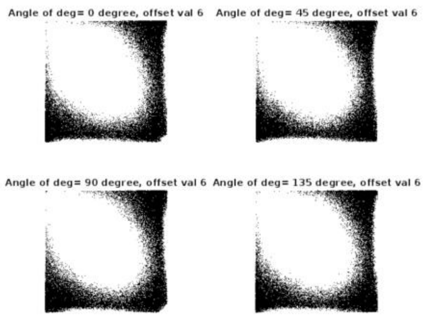


Figure 10: co-ocurrence matrix for offset value 6 and four different degrees.

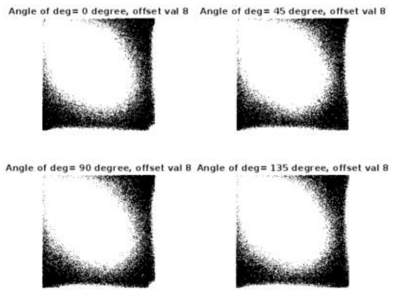
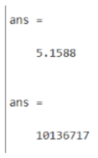


Figure 11: co-ocurrence matrix for offset value 8 and four different degrees.

6) Plotting the values of the four different offset at four different angles.

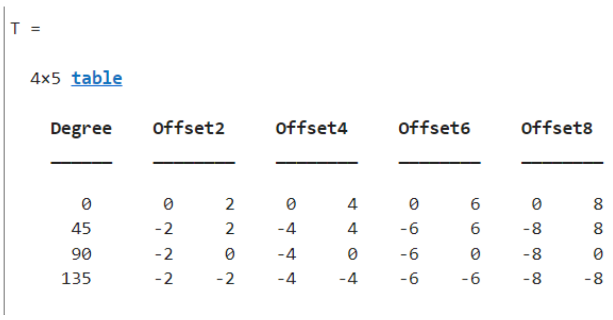


```
ans =

    5.1588


ans =

    10136717
```



```
T =

  4x5 table

    Degree    Offset2     Offset4     Offset6     Offset8
    ______    ________    ________    ________    ________

       0       0     2     0     4     0     6     0     8
      45      -2     2    -4     4    -6     6    -8     8
      90      -2     0    -4     0    -6     0    -8     0
     135      -2    -2    -4    -4    -6    -6    -8    -8
```

Figure 12: Plotting the values and four different offsets at four different angles

## Task 2.2 – IMPLEMENTATION STEPS FOR GRAY CO-OCCURRENCE MATRIX.

The given image is read and then estimates its co-occurrence matrix. And in the co-occurrence matrix is has given the offset parameter as 1 and estimated the co-occurrence matrix with four different degrees they are : 0 degree, 45 degree, 90 degree and 135 degree. In addition to the gray co-occurrence properties the two features are added from the histograms of 1st order derivatives and total 6 features were added and they are- energy, correlation, contrast, homogeneity, Angular second moment and inverse difference moment which are selected from the Spectral approach method features.

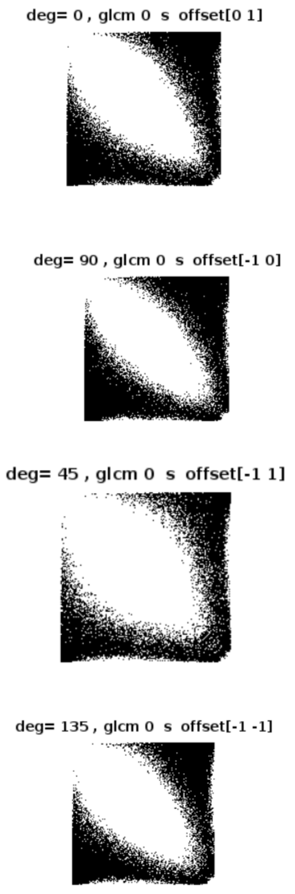


Figure 13: gray co-occurrence matrix for four different angles with offset parameter as 1.

The Angular second moment features values for four different degrees are:
At 0 degree= 2.129623268091494e+16
At 45 degree= 2.063643953751654e+16
At 90 degree= 2.106627912132198e+16
At 135 degree= 2.063320342960538e+16

The inverse difference moment features values for four different degrees are:
At 0 degree= 3.735866572800000e+10
At 45 degree= 3.677539532800000e+10
At 90 degree= 3.715642163200000e+10
At 135 degree= 3.677251174400000e+10

## Task 2.3 – IMPLEMENTATIONS STEPS FOR GRAY LEVEL RUN LENGTH (GLRL)

In GLRL feature, the bit size given is =16. And used the five features of GLRL they are : short run emphasis, Long run emphasis , high gray level run emphasis, low gray level run emphasis, gray level non-uniformity.
The feature values obtained for the bit size of an image is 16 they are :

```
ShortRE1 =0.7875
LongRE1=3.3095
GrayLevelNon_1 =1524.4251
RunPer1 =0.6843
RunLN1 =6459.8029
LowGREY_level1 =0.0165
High_gray_level_run1= 92.4133
```

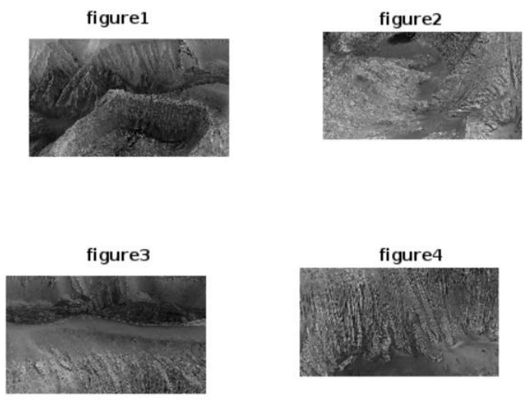


Figure 14: The bit size given for the image is 16.

## Reference

**[1].** Texture Analysis lecture, University of Lincoln blackboard, computer vision week 5 lecture. [Prof. Luc Bidaut]

**[2].** Haralick, R.M., Shanmugam, K. and Dinstein, I.H., 1973. Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, (6), pp.610-62

# Task 3: Object Tracking

***Abstract*— In this assessment, csv read data files are provided in which x and y csv files containing the real files of the moving target and the a and b csv files containing the noisy version which are provided by the generic video generator and object tracking is done by implementing the Kalman filter taking constant time intervals Δt = 0.1 and covariance matrixes Q and R of the noises is provided. And plotting is done for both the estimated trajectory of coordinates [x*,y*] and with the real [x,y] and the noisy ones [a,b] for comparison. The quality of tracking is assessed by calculating the mean and standard deviation of absolute error and the root mean squared error by comparing both noisy and estimated coordinates to the ground truth.**

## INTRODUCTION

**Object Tracking:** It is the task of taking the set of object detection and creating their unique ID for every detections and tracking the every objects while moving around the frames in the video [1]. OpenCv is popular method for for the object tracking in the real scenerios.

**Kalman filter:** One of the most essential and widely used estimate methods is the Kalman Filter. Based on faulty and uncertain data, the Kalman Filter generates hidden variable estimations. In addition, the Kalman Filter predicts future system states based on previous estimates [1].

Predict
$$\boxed{\begin{aligned}\hat{\boldsymbol{x}}_k &= \boldsymbol{F}\boldsymbol{x}_{k-1}\\ \hat{\boldsymbol{P}}_k &= \boldsymbol{F}\boldsymbol{P}_{k-1}\boldsymbol{F}^T + \boldsymbol{Q}\end{aligned}}$$

Update
$$\boxed{\begin{aligned}\boldsymbol{x}_k &= \hat{\boldsymbol{x}}_k + \boldsymbol{K}(\boldsymbol{z}_k - \hat{\boldsymbol{z}}_k)\\ \boldsymbol{P}_k &= \hat{\boldsymbol{P}}_k - \boldsymbol{K}_k\boldsymbol{S}_k\boldsymbol{K}_k^T\end{aligned}}$$

Where $\boldsymbol{x}$ = means state vector (the hat ^ means as “prediction”) , $\boldsymbol{F}$ = means matrix of the motion model, $\boldsymbol{Q}$ = means matrix of the motion noise , $\boldsymbol{P}$ = means covariance matrix of x (i.e. when it is in uncertainty)

Figure: Mathematical function of Kalman filter predict and update [1].

**Validation Gating:** For determining the validation region, the maximum distance of the target that has traveled is considered for the previous time step if it is moving at the maximum speed [1].

For better solution for computing the mahalanobis distance is considered for the observation of I and target j and checked the value with the threshold whether it is less or not.If the distance traveled is greater than the threshold value then it is considered as outlier and not used to update the Kalman filter for the observation [1].

**SADANAND VITHAL GIRADDI**
**ID – 25895786**
**Computer Vision Assignment**

$$\boxed{(\mathbf{z}_i - \hat{\mathbf{z}}_j)^T\, \mathbf{S}_{ij}^{-1}\, (\mathbf{z}_i - \hat{\mathbf{z}}_j) \leqslant \lambda}$$

($\lambda$ = 9.21 for 2D observations)

Fig: mathematical function of validation gating [1].

**For Evaluation:** By checking check on the plot, computing mean and the standard deviation values can be measured by the quality of the tracking of the error,

defined as : $ei = \sqrt{}(xi - pxi)\ 2 + (yi - pyi)\ 2$ [1]

Also, the Root Mean Squared error(RMS) is the measure of tracking accuracy:

$RMS = \sqrt{}\ \sum ei\ N\ 2\ i{=}1$ � [1]

## IMPLEMENTATIONS OF STEPS WITH RESULTS

1) Importing all the data files provided in the assessment using the csvread function of MatLab.
2) The files x.csv and y.csv contains the real coordinates of the target trajectories. And the file a.csv and b.csv contains the noisy version coordinates.
3) At first, the mean and standard deviation of respective noises (nx =a-x) and (ny-b-y) were computed and plotted the two trajectories by giving two different color commands of the MatLab function. And their histograms are verified for zero-mean Gaussian noises.

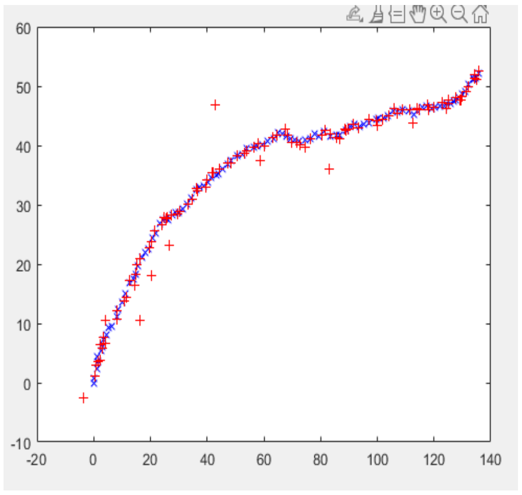


Figure 1: initial plotting the two trajectories.

Initial Mean values of two trajectories
nx= 0.0125
ny= -0.1202
Initial standard deviation values of two trajectories
std (nx)= 1.5449
std (ny)= 1.6037

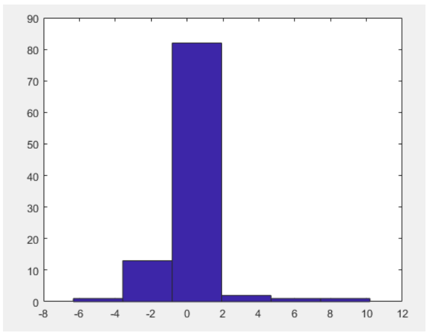


Figure 2: Histogram of nx.

4) Kalman filter is implemented and predicted for the noisy coordinate of [a,b] is taken as inputs and produced the outputs for the estimated coordinates [x*,y*] by using the constant velocity motion model F with the constant time intervals of Δt = 0.1 and the Cartesian observation model H. The covariance matrices Q and R for the respective noises are as follows: $Q$ = [ 0.16 0 0 0 0 0.36 0 0 0 0 0.16 0 0 0 0 0.36] $R$ = [ 0.25 0 0 0.25]. with the help of Kalman predict the MatLab function is provided for predicting the state and state covariance of the model. And with the help of Kalman update is done for updating the estimated state and estimated covariance of the model without adding the gating is done earlier and later in the result we will find Kalman update is done by adding the validation gating. And at the same time the Kalman Tracking function implements the whole Kalman filter which is based on tracker for the target moving according to the constant velocity of the model F with the constant time intervals of Δt = 0.1 and the covariance matrices Q and R for the respective noises are as follows: $Q$ = [ 0.16 0 0 0 0 0.36 0 0 0 0 0.16 0 0 0 0 0.36] $R$ = [ 0.25 0 0 0.25].

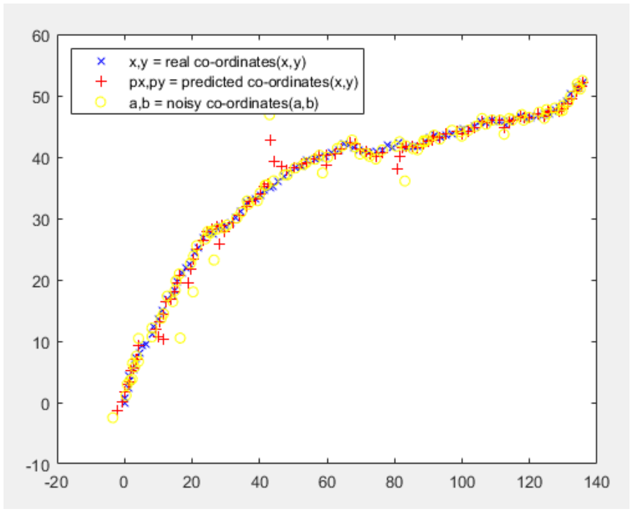


Figure 3: The final comparison of both noisy and estimated coordinates of ground truth.

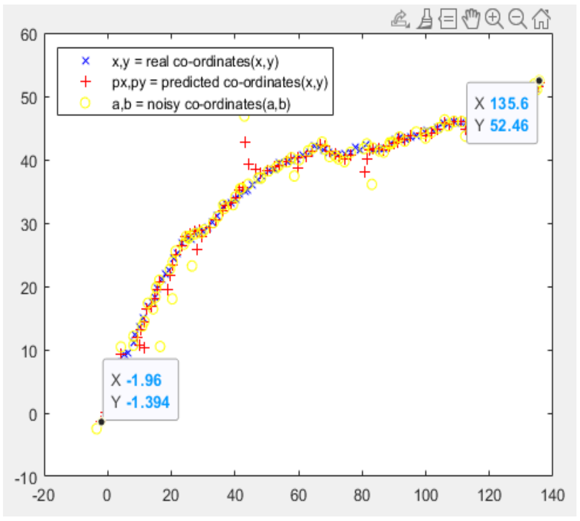


Figure 4: Plotted with the values for the final comparison of both noisy and estimated coordinates of ground truth.

5) And now the Kalman function is updated with the validation gating and the Kalman filter is validated using gating. This gating is used for getting rid of an observation which are unlikely generated by the target. For Validation gating > 20 is considered.

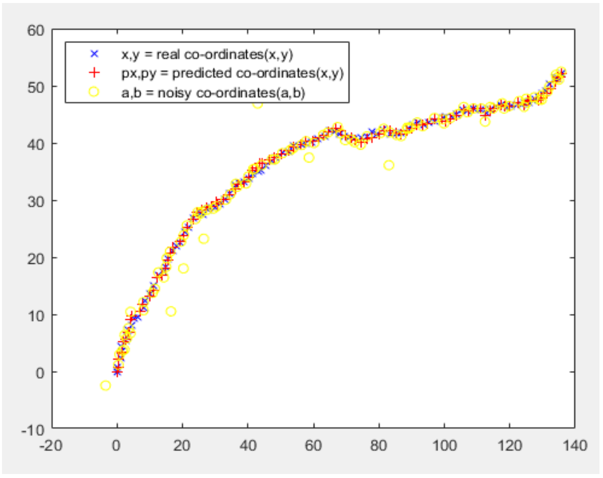


Figure 5:Updated with validation gating for the final comparision of both noisy and estimated coordinates of ground truth.

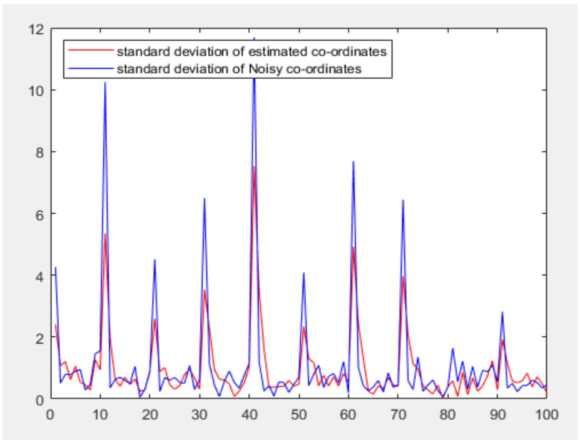


Figure 6: Plot for comparing both noisy and estimated co-ordinates of ground truth values.

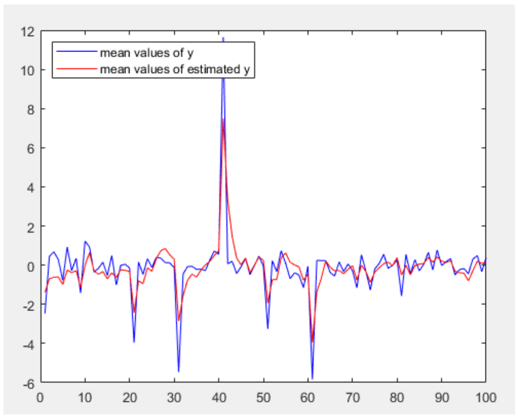


Figure 7: mean values of y and mean values of estimated y.

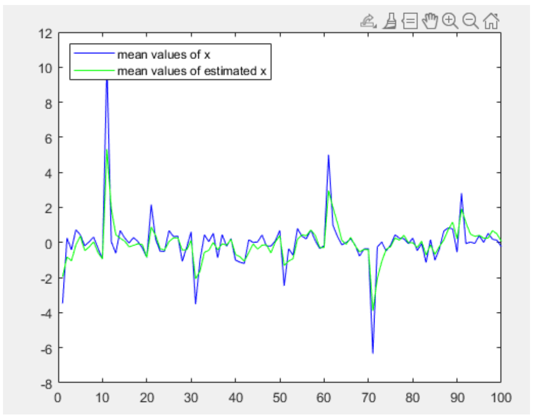


Figure 10: mean values of x and mean values of estimated x.

**6) Calculating the mean and standard deviation of absolute error and Root mean squared error for estimated value**

```
Standard_deviation_absolute_error =
  Columns 1 through 10
    2.4053    1.0749    1.2108    0.6175    1.0551    0.5304    0.4560
0.2869    1.2670    0.9367
  Columns 11 through 20
    5.3628    1.9391    0.6454    0.4075    0.7004    0.4651    0.6473
0.2465    0.2898    0.904
  Columns 21 through 30
    2.5908    0.8866    1.0102    0.4537    0.3128    0.4222    0.7920
0.9588    0.6668    0.3174
  Columns 31 through 40
    3.5283    2.2580    0.9481    0.6356    0.6134    0.4853    0.0758
0.2496    0.5256    1.0061
  Columns 41 through 50
    7.5213    3.4893    1.5893    0.3723    0.3821    0.3957    0.4087
0.5971    0.4130    0.4712
  Columns 51 through 60
    2.3389    1.2956    1.1722    0.4262    0.7607    0.4148    0.7137
0.4155    0.8191    0.5450
  Columns 61 through 70
    4.9231    2.4130    1.3064    0.2530    0.1498    0.4194    0.3280
0.6874    0.4536    0.4055
  Columns 71 through 80

3.9725    2.0580    1.1355    0.9655    0.4214    0.2847    0.1545
0.4418    0.0266    0.3898
Columns 81 through 90
    0.5810    0.0770    0.8634    0.1380    0.6768    0.2507    0.4129
0.7438    1.2331    0.2923
  Columns 91 through 100
    1.9200    1.1469    0.5782    0.5190    0.5812    0.8339    0.3839
0.7147    0.4934    0.1747
```

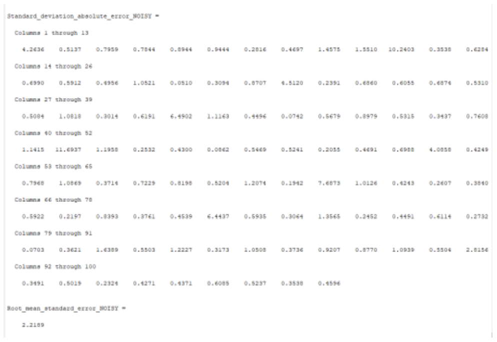
**Root_mean_standard_error =**
**1.5204**

**7) Calculating the mean and standard deviation of absolute error and Root mean squared error for Noisy value**

```
Standard_deviation_absolute_error_NOISY =
  Columns 1 through 13
    4.2636    0.5137    0.7959    0.7844    0.8944    0.9444    0.2816
0.4697    1.4575    1.5510   10.2403    0.3538    0.6284
  Columns 14 through 26
    0.6990    0.5912    0.4956    1.0521    0.0510    0.3094    0.8707
4.5120    0.2391    0.6860    0.6055    0.6874    0.5310
  Columns 27 through 39
    0.5084    1.0818    0.3014    0.6191    6.4902    1.1163    0.4496
0.0742    0.5679    0.8979    0.5315    0.3437    0.7608
  Columns 40 through 52
    1.1415   11.6937    1.1958    0.2532    0.4300    0.0862    0.5469
0.5241    0.2055    0.4691    0.6988    4.0858    0.4249
  Columns 53 through 65
    0.7968    1.0869    0.3714    0.7229    0.8198    0.5204    1.2074
0.1942    7.6873    1.0126    0.4243    0.2607    0.3840
  Columns 66 through 78
    0.5922    0.2197    0.8393    0.3761    0.4539    6.4437    0.5935
0.3064    1.3565    0.2452    0.4491    0.6114    0.2732
  Columns 79 through 91

    0.0703    0.3621    1.6389    0.5503    1.2227    0.3173    1.0508
0.3736    0.9207    0.8770    1.0939    0.5504    2.8156
  Columns 92 through 100
    0.3491    0.5019    0.2324    0.4271    0.4371    0.6085    0.5237
0.3538    0.4596
```

**Root_mean_standard_error_NOISY =**
**2.2189**



```
Standard_deviation_absolute_error_NOISY =

  Columns 1 through 13

    4.2636    0.5137    0.7959    0.7844    0.8944    0.9444    0.2816    0.4697    1.4575    1.5510   10.2403    0.3538    0.6284

  Columns 14 through 26

    0.6990    0.5912    0.4956    1.0521    0.0510    0.3094    0.8707    4.5120    0.2391    0.6860    0.6055    0.6874    0.5310

  Columns 27 through 39

    0.5084    1.0818    0.3014    0.6191    6.4902    1.1163    0.4496    0.0742    0.5679    0.8979    0.5315    0.3437    0.7608

  Columns 40 through 52

    1.1415   11.6937    1.1958    0.2532    0.4300    0.0862    0.5469    0.5241    0.2055    0.4691    0.6988    4.0858    0.4249

  Columns 53 through 65

    0.7968    1.0869    0.3714    0.7229    0.8198    0.5204    1.2074    0.1942    7.6873    1.0126    0.4243    0.2607    0.3840

  Columns 66 through 78

    0.5922    0.2197    0.8393    0.3761    0.4539    6.4437    0.5935    0.3064    1.3565    0.2452    0.4491    0.6114    0.2732

  Columns 79 through 91

    0.0703    0.3621    1.6389    0.5503    1.2227    0.3173    1.0508    0.3736    0.9207    0.8770    1.0939    0.5504    2.8156

  Columns 92 through 100

    0.3491    0.5019    0.2324    0.4271    0.4371    0.6085    0.5237    0.3538    0.4596

Root_mean_standard_error_NOISY =

    2.2189
```

Figure 9: Root mean standard error noisy values

## References

[1]. Visual tracking 2 lecture and its workshop, University of Lincoln blackboard, computer vision week 2 lecture. [Dr Wenting Duan]

# APPENDIX

**Task 1 code: For image segmentation and detection.**

**Main code:**

```
close all
clc
%reading the files of skin lesion dataset of org images and GT images
Org_folder = dir('org data\*.jpg');
GT_folder=dir('GT\*.png');
for i=1:numel(Org_folder)
   basefile = Org_folder(i);
   fullfilename = fullfile(basefile.folder, basefile.name);
   file_segment = GT_folder(i);

   disp(fullfilename);
   filenamesegment=fullfile(file_segment.folder,file_segment.name);
   segmentimages=imread(filenamesegment);
   segment=logical(segmentimages);
   I = imread(fullfilename);
   Grayscaleimage=rgb2gray(I)
%     subplot(1,3,2)
%     imshow(Grayscaleimage)

   Binaryimage= im2bw(Grayscaleimage)
   compliment=imcomplement(Binaryimage)
%     subplot(1,3,3)
   imshow(compliment)
   impixelinfo
   imwrite(compliment,basefile.name)
   k(i)=dice(compliment,segment)
end
mean_value=mean(k)
std_devia=std(k)
bar(k)
Org_folder = Org_folder(~[Org_folder.isdir]);
Org_folder = {Org_folder.name}
dicescore1= categorical(Org_folder);
dice_similarity1= categorical(k(i))
bar(dicescore1,k())
xlabel('Total_Number_of_images')
ylabel('DICE-SCORES')
```

## TASK 2 code : Feature calculation

**Main code of task 2.1:**

**This code is for selecting the features for both the radius and the direction and giving the four different values of offset and tested with four different angles.**

```
clear all
clc
a = imread('ImgPIA.jpg');
figure;
imshow(a);
a = rgb2gray(a);
title('our ImgPIA image')

%transform of our ImgPIA image
B = fft2(a,256,256);
figure;
imshow(B)

% SEGMENTing our ImgPIA INTO four different SECTION
I = a;
a1=I(1:size(I,1)/2,1:size(I,2)/2,:);subplot(2,2,1);imshow(a1);
subplot(2,2,1);
title('figure1')
a2=I(size(I,1)/2+1:size(I,1),1:size(I,2)/2,:);subplot(2,2,2);imshow(a2);
subplot(2,2,2);
title('figure2')
a3=I(1:size(I,1)/2,size(I,2)/2+1:size(I,2),:);subplot(2,2,3);imshow(a3);
subplot(2,2,3);
title('figure3')
a4=I(size(I,1)/2+1:size(I,1),size(I,2)/2+1:size(I,2),:);subplot(2,2,4);imshow(a4);
subplot(2,2,4);
title('figure4')

%using fourier transform
B = fft2(a,256,256);
figure;
imshow(B);
B = fftshift(B); % using discrete fourier transform function for four sections
figure;
imshow(B);

% Measuring the min and max value of the transform amplitudes.
min(min(abs(B)))
max(max(abs(B)))
figure;
imshow(abs(B),[0 100]); colormap(jet); colorbar
figure;
imshow(log(1+abs(B)),[0,3]); colormap(jet); colorbar
```

```
% Looking at the phases
figure;
imshow(angle(B),[-pi,pi]); colormap(jet); colorbar

%now using inverse of fourier transform
I = ifft2(B,256,256);
figure;
imshow(I);

% using the fourier with inverse function together
B = fft2(a,256,256);
figure;
imshow(B);
B = fftshift(B); % using discrete fourier transform function for four sections
figure;
imshow(B);

% Measuring the min and max value of the transform amplitudes.
min(min(abs(B)))
max(max(abs(B)))
figure;
imshow(abs(B),[0 100]); colormap(jet); colorbar
figure;
imshow(log(1+abs(B)),[0,3]); colormap(jet); colorbar
% Looking for phases
figure;
imshow(angle(B),[-pi,pi]); colormap(jet); colorbar

B = fft2(a,256,256);
figure;
imshow(B);

I = ifft2(B,256,256);
figure;
imshow(I);

% TASK 2 part 1:  selecting the features for both radius and direction
% which is
%described in the Spectral Approach session from the Feature Extraction
%lecture.

figure;
imshow(a);
figure;
%%offset value 2,4,6,8
% My offset =2
offsets = [0 2];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,1);imshow(co_curmatrix);
title('Angle of deg= 0 degree, offset val 2')

offsets = [-2 2];
```

```
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,2);imshow(co_curmatrix);
title('Angle of deg= 45 degree, offset val 2')

offsets = [-2 0];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,3);imshow(co_curmatrix);
title('Angle of deg= 90 degree, offset val 2')

offsets = [-2 -2];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,4);imshow(co_curmatrix);
title('Angle of deg= 135 degree, offset val 2')


figure;
% My offset =4
offsets = [0 4];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,1);imshow(co_curmatrix);
title('Angle of deg= 0 degree, offset val 4')

offsets = [-4 4];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,2);imshow(co_curmatrix);
title('Angle of deg= 45 degree, offset val 4')

offsets = [-4 0];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,3);imshow(co_curmatrix);
title('Angle of deg= 90 degree, offset val 4')

offsets = [-4 -4];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,4);imshow(co_curmatrix);
title('Angle of deg= 135 degree, offset val 4')

figure;
% My offset =6
offsets = [0 6];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,1);imshow(co_curmatrix);
title('Angle of deg= 0 degree, offset val 6')

offsets = [-6 6];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,2);imshow(co_curmatrix);
title('Angle of deg= 45 degree, offset val 6')

offsets = [-6 0];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,3);imshow(co_curmatrix);
```

```
title('Angle of deg= 90 degree, offset val 6')

offsets = [-6 -6];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,4);imshow(co_curmatrix);
title('Angle of deg= 135 degree, offset val 6')


figure;
% My offset =8
offsets = [0 8];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,1);imshow(co_curmatrix);
title('Angle of deg= 0 degree, offset val 8')

offsets = [-8 8];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,2);imshow(co_curmatrix);
title('Angle of deg= 45 degree, offset val 8')

offsets = [-8 0];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,3);imshow(co_curmatrix);
title('Angle of deg= 90 degree, offset val 8')

offsets = [-8 -8];
co_curmatrix = graycomatrix(a, 'offset', offsets,'NumLevels', 256, 'Symmetric',true);
subplot(2,2,4);imshow(co_curmatrix);
title('Angle of deg= 135 degree, offset val 8')


% plotting the values = ["Degree";"Offset1";"Offset2";"Offset3";"Offset4"];

Degree = [0;45;90;135];
Offset2 = [0 2; -2 2; -2 0; -2 -2];
Offset4 = [0 4; -4 4; -4 0; -4 -4];
Offset6 = [0 6; -6 6; -6 0; -6 -6];
Offset8 = [0 8; -8 8; -8 0; -8 -8];
%T = table("Degree";"Offset2";"Offset4";"Offset6";"Offset8")
T = table(Degree,Offset2,Offset4,Offset6, Offset8)
%writetable(T,'offsetvaltest.xlsx','Sheet',1);
```

**Main code of task 2,2:**

**This code is for selecting the six different features from spectral approach method and evaluating the values.**

```
close all
clc
%features =[]
a = imread('ImgPIA.jpg');
figure;
```

```
b=imhist(a);
imshow(a);
a = rgb2gray(a);
title('our ImgPIA image')

glcm_0_s = graycomatrix(a, 'offset', [0 1],'NumLevels', 256, 'Symmetric', true);
stats_0_s = graycoprops(glcm_0_s,{'contrast','correlation','energy','homogeneity'});
Angsecondmoment0deg = Angularsecondmomentum(glcm_0_s);
invdiffmoment0deg = inversedifferencemoment(glcm_0_s);
figure;
subplot(2,2,1);
imshow(glcm_0_s);
title('deg= 0 , glcm_ 0 _ s _ offset[0 1]')
%title('Angle of deg= 0 degree, offset val 2')


%
%
% glcm_0_ns = graycomatrix(a, 'offset', [0 1], 'NumLevels', 256,'Symmetric', false);
% stats_0_ns = graycoprops(glcm_0_ns,{'contrast','correlation','energy','homogeneity'});
% Angsecondmoment0deg = Angularsecondmomentum(glcm_0_ns);
% invdiffmoment0deg = inversedifferencemoment(glcm_0_ns);
% figure;
% subplot(2,2,1);
% imshow(glcm_0_ns);
% title('deg= 0, glcm_ 0 _ ns _ offset[0 1]')
%



glcm_45_s = graycomatrix(a, 'offset', [-1 1],'NumLevels', 256, 'Symmetric', true);
stats_45_s = graycoprops(glcm_45_s,{'contrast','correlation','energy','homogeneity'});
Angsecondmoment45deg = Angularsecondmomentum(glcm_45_s);
invdiffmoment45deg = inversedifferencemoment(glcm_45_s);
figure;
subplot(2,2,2);
imshow(glcm_45_s);
title('deg= 45 , glcm_ 0 _ s _ offset[-1 1]')
%title('Angle of deg= 0 degree, offset val 2')


%
% glcm_45_ns = graycomatrix(a, 'offset', [-1 1], 'NumLevels', 256,'Symmetric', false);
% stats_45_ns = graycoprops(glcm_45_ns,{'contrast','correlation','energy','homogeneity'});
% Angsecondmoment45deg = Angularsecondmomentum(glcm_45_ns);
% invdiffmoment45deg = inversedifferencemoment(glcm_45_ns);
% figure;
% subplot(2,2,2);
% imshow(glcm_45_ns);
% title('deg= 45, glcm_ 0 _ ns _ offset[-1 1]')
```

```
glcm_90_s = graycomatrix(a, 'offset', [-1 0],'NumLevels', 256, 'Symmetric', true);
stats_90_s = graycoprops(glcm_90_s,{'contrast','correlation','energy','homogeneity'});
Angsecondmoment90deg = Angularsecondmomentum(glcm_90_s);
invdiffmoment90deg = inversedifferencemoment(glcm_90_s);
figure;
subplot(2,2,3);
imshow(glcm_90_s);
title('deg= 90 , glcm_ 0 _ s _ offset[-1 0]')
%title('Angle of deg= 0 degree, offset val 2')

%
%
% glcm_90_ns = graycomatrix(a, 'offset', [-1 0], 'NumLevels', 256,'Symmetric', false);
% stats_90_ns = graycoprops(glcm_90_ns,{'contrast','correlation','energy','homogeneity'});
% Angsecondmoment90deg = Angularsecondmomentum(glcm_90_ns);
% invdiffmoment90deg = inversedifferencemoment(glcm_90_ns);
% figure;
% subplot(2,2,3);
% imshow(glcm_90_ns);
% title('deg= 90, glcm_ 0 _ ns _ offset[-1 0]')
%



glcm_135_s = graycomatrix(a, 'offset', [-1 -1],'NumLevels', 256, 'Symmetric', true);
stats_135_s = graycoprops(glcm_135_s,{'contrast','correlation','energy','homogeneity'});
Angsecondmoment135deg = Angularsecondmomentum(glcm_135_s);
invdiffmoment135deg = inversedifferencemoment(glcm_135_s);
figure;
subplot(2,2,3);
imshow(glcm_135_s);
title('deg= 135 , glcm_ 0 _ s _ offset[-1 -1]')
%title('Angle of deg= 0 degree, offset val 2')



%
% glcm_135_ns = graycomatrix(a, 'offset', [-1 -1], 'NumLevels', 256,'Symmetric', false);
% stats_135_ns = graycoprops(glcm_135_ns,{'contrast','correlation','energy','homogeneity'});
% Angsecondmoment135deg = Angularsecondmomentum(glcm_135_ns);
% invdiffmoment135deg = inversedifferencemoment(glcm_135_ns);
% figure;
% subplot(2,2,3);
% imshow(glcm_135_ns);
% title('deg= 135, glcm_ 0 _ ns _ offset[-1 -1]')
%
```

**Function:**

**Adding the Angular second moment feature**

```
function [sum]=Angularsecondmomentum(gc)

[x,y]=size(gc);
sum=0;
for i=1:x
    for j=1:y
        sum=sum+(gc(x,y))^2;
    end
end
```

**Adding the inverse difference moment feature**

```
function [sum1]=inversedifferencemoment(gc)

[x,y]=size(gc);
sum1=0;
for i=1:x
    for j=1:y
        sum1=sum1 + (1/(1+(x-y)^2))*gc(x,y);
    end
end
```

**The below code is for selecting the five features from gray level run lenth (GLRL) matrix**

```
clear all

clc

a = imread('ImgPIA.jpg');

figure

imshow(a)


a = rgb2gray(a);

title('OriginalImage')

Im = a;

a1=Im(1:size(Im,1)/2,1:size(Im,2)/2,:);subplot(2,2,1);imshow(a1);

subplot(2,2,1);

title('figure1')

a2=Im(size(Im,1)/2+1:size(Im,1),1:size(Im,2)/2,:);subplot(2,2,2);imshow(a2);

subplot(2,2,2);
```

```
title('figure2')
a3=Im(1:size(Im,1)/2,size(Im,2)/2+1:size(Im,2),:);subplot(2,2,3);imshow(a3);
subplot(2,2,3);
title('figure3')
a4=Im(size(Im,1)/2+1:size(Im,1),size(Im,2)/2+1:size(Im,2),:);subplot(2,2,4);imshow(a4);
subplot(2,2,4);
title('figure4')

image1= a4 %imcrop(image);
image2=image1(1:128,1:128);
image2=double(image2);
[e,f]=size(image2);

% --------- bit-depth image------------------
bit_size = 16; % 4 bits - 16, 6 bits - 32, 8 bits - 64,

Image_max=max(max(image2));
Image_min=min(min(image2));
newimage=image2-Image_min;
New_max=max(max(newimage));
New_min=min(min(newimage));
Q=round(New_max/bit_size);
[e,f]=size(newimage);
Quant=0;
for i=1:e
    for j=1:f
        Im = newimage(i,j);
        for B = 1:bit_size
            if (Im>Quant)&(Im<=Quant+Q)
                newimage(i,j)=B/bit_size;
                Quant=Quant+Q;
            end
        end
    end
end
newim_max=max(max(newimage));
newim_1=newimage/newim_max;
newim_2=round(newim_1*bit_size)+1;
dir=0;
dist1=1;
if (dir == 1)
    newim_2=newim_2';
end
cx = max(max(newim_2));
cn = min(min(newim_2));
hl = (cx-cn)+1;
[k,l] = size(newim_2);
f=k*l;
count=1;
e=1;
col=1;
glrl(cx,k)=0;
maxcount(k*l)=0;
mc=0;

%--------------------using the glrl matrix---------------
for j=1:k
    for k=1:l-dist1
        mc=mc+1;
        g=newim_2(j,k);
        a=newim_2(j,k+dist1);
        if (g==a)&(g~=0)
            count=count+1;
            e=count;
            col=count;
            maxcount(mc)=count;
        else glrl(g,e)=glrl(g,e)+1;col=1;
            count=1;
            e=1;
        end
    end
    glrl(a,col)=glrl(a,col)+1;
    count=1;
    e=1;
end
Im=(cx:cn);
e=glrl(cn:cx,:);
m1=e';
maxrun=max(max(maxcount));
S=0;
G(hl)=0;
R(l)=0;
for u=1:hl
    for v=1:l
        G(u)=G(u)+e(u,v);
        S=S+e(u,v);
    end
end
for u1=1:l
    for v1=1:hl
        R(u1)=R(u1)+m1(u1,v1);
    end
end
[dim,dim1]=size(G);
Short_Run_Emphasis=0;     LRE=0;     Gray_level_NonUniformity=0;     RLN=0;     Run_Percentage=0;
low_gray_level_run_emphasis=0; high_gray_level_run_emphasis=0;
```

```
for h1= 1:maxrun
    Short_Run_Emphasis=Short_Run_Emphasis+(R(h1)/(h1*h1));
    LRE=LRE+(R(h1)*(h1*h1));
    RLN=RLN+(R(h1)*R(h1));
    Run_Percentage=Run_Percentage+R(h1);
end
ShortRE1=Short_Run_Emphasis/S;
LongRE1=LRE/S;
RunLN1=RLN/S;
RunPer1=Run_Percentage/f;
for h2=1:hl
    Gray_level_NonUniformity=(Gray_level_NonUniformity+G(h2)^2);
    low_gray_level_run_emphasis=low_gray_level_run_emphasis+(G(h2)/(h2*h2));
    high_gray_level_run_emphasis=high_gray_level_run_emphasis+(h2*h2)*G(h2);
end
GrayLevelNon_1=Gray_level_NonUniformity/S;
LowGREY_level1=low_gray_level_run_emphasis/S;
High_gray_level_run1=high_gray_level_run_emphasis/S;
clc

% --------------------------DISPLAY THE PARAMETERS-----------------------



disp(sprintf('%6.4f',ShortRE1))
disp(sprintf('%6.4f',LongRE1))
disp(sprintf('%6.4f',GrayLevelNon_1))
disp(sprintf('%6.4f',RunPer1))
disp(sprintf('%6.4f',RunLN1))
disp(sprintf('%6.4f',LowGREY_level1))
disp(sprintf('%6.4f',High_gray_level_run1))
```

**TASK 3 code: Object Tracking**

**Main code:**

```
% x and y csv read files contain the real co-ordinates
% a and b csv files contain Noisy version which are provided by the generic
% video detector.
x=csvread('x.csv');
y=csvread('y.csv');
a=csvread('a.csv');
b=csvread('b.csv');

%ploting the values from x and y csv files
plot(x, y, 'xb');
hold;
plot(a, b, '+r');

nx = a-x;
ny = b-y;
mean(nx)
mean(ny)
std(nx)
std(ny)


% plotting histogram figure for nx and ny
histogram(nx,6)
histogram(ny,6)

%plotting
plot(x, y, 'xb');
hold;
plot(a, b, '+r');
title('noise graph')

%task2: using kalman tracking
z=[a;b]
[px, py] = kalmanTracking(z)

nnx = px-x;
nny = py-y;
mean(nx)
mean(ny)
plot(x, y, 'xb');
hold on
plot(a, b, '+r');
hold off

plot(x,y, 'xb', px, py, '+r', a,b,'oy');
legend({'x,y = real co-ordinates(x,y)','px,py = predicted co-ordinates(x,y)','a,b = noisy co-ordinates(a,b)'},
'Location', 'northwest')
```

```
% Calculating the mean and standard deviation of absolute error and Root mean squared error for estimated value

Standard_deviation_absolute_error = sqrt( ((x-px).^2) + ((y-py).^2) )
Root_mean_standard_error = rms(Standard_deviation_absolute_error)

% Calculating the mean and standard deviation of absolute error and Root mean squared error for Noisy Co-ordinates value

Standard_deviation_absolute_error_NOISY = sqrt( ((x-a).^2) + ((y-b).^2) )
Root_mean_standard_error_NOISY = rms(Standard_deviation_absolute_error_NOISY)
```

**<u>Function codes</u>**:

1) <u>Kalman predict</u>

```
function [xp, Pp] = kalmanPredict(x, P, F, Q)
% Prediction step of Kalman filter.
% x: state vector
% P: covariance matrix of x
% F: matrix of motion model
% Q: matrix of motion noise
% Return predicted state vector xp and covariance Pp
xp = F * x; % predict state
Pp = F * P * F' + Q; % predict state covariance
end
```

2) <u>Kalman Update without gating</u>

```
function [xe, Pe] = kalmanUpdate(x, P, H, R, z)
% Update step of Kalman filter.
% x: state vector
% P: covariance matrix of x
% H: matrix of observation model
% R: matrix of observation noise
% z: observation vector
% Return estimated state vector xe and covariance Pe
S = H * P * H' + R; % innovation covariance
K = P * H' * inv(S); % Kalman gain
zp = H * x; % predicted observation
%%%%%%%%% UNCOMMENT FOR VALIDATION GATING %%%%%%%%%%
%gate = (z - zp)' * inv(S) * (z - zp);
%if gate > 9.21
% warning('Observation outside validation gate');
% xe = x;
% Pe = P;
% return
%end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
xe = x + K * (z - zp); % estimated state
Pe = P - K * S * K'; % estimated covariance
end
```

3) <u>Kalman tracking</u>

```
function [px, py] = kalmanTracking(z)
% Track a target with a Kalman filter
% z: observation vector
% Return the estimated state position coordinates (px,py)
dt = 0.1; % time interval
N = length(z); % number of samples
F = [1 dt 0 0; 0 1 0 0; 0 0 1 dt; 0 0 0 1]; % CV motion model
Q = [0.16  0 0 0; 0 0.36 0 0; 0 0 0.16 0; 0 0 0 0.36]; % motion noise
H = [1 0 0 0; 0 0 1 0]; % Cartesian observation model
R = [0.25 0; 0 0.25]; % observation noise
x = [0 0 0 0]'; % initial state
P = Q; % initial state covariance
s = zeros(4,N);
for i = 1 : N
[xp, Pp] = kalmanPredict(x, P, F, Q);
[x, P] = gateway_code(xp, Pp, H, R, z(:,i));
s(:,i) = x; % save current state
end
px = s(1,:); % NOTE: s(2, :) and s(4, :), not considered here,
py = s(3,:); % contain the velocities on x and y respectively
end
```

4) <u>Plotting mean values of x and estimated x*</u>

```
plot(nx, 'b')
hold on
plot( nnx, 'g')
hold off
legend({'mean values of x','mean values of estimated x'}, 'Location', 'northwest')
```

5) <u>Plotting mean values of y and estimated y*</u>

```
plot(ny,'b')
hold on
plot( nny, 'r')
legend({'mean values of y','mean values of estimated y'}, 'Location', 'northwest')
hold off
```

6) <u>Plotting standard deviation absolute error and with noisy ones.</u>

```
plot(Standard_deviation_absolute_error,'r')
%legend('standard deviation of estimated co-ordinates')
hold on
plot(Standard_deviation_absolute_error_NOISY,'b')
%egend('standard deviation of Noisy co-ordinates')
```

```
legend({'standard deviation of estimated co-ordinates','standard deviation of Noisy co-ordinates'}, 'Location',
'northwest')
hold off
```

7) Kalman update with validation gating.

```
function [xe, Pe] = kalmanUpdate(x, P, H, R, z)
% Update step of Kalman filter.
% x: state vector
% P: covariance matrix of x
% H: matrix of observation model
% R: matrix of observation noise
% z: observation vector
% Return estimated state vector xe and covariance Pe
S = H * P * H' + R; % innovation covariance
K = P * H' * inv(S); % Kalman gain

zp = H * x; % predicted observation
%%%%%%%%% UNCOMMENT FOR VALIDATION GATING %%%%%%%%%%
gate = (z - zp)' * inv(S) * (z - zp);
if gate > 20
warning('Observation outside validation gate');
xe = x;
Pe = P;
return
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
xe = x + K * (z - zp); % estimated state
Pe = P - K * S * K'; % estimated covariance
end
```