ECE 7410/ENGI 9804 IMAGE PROCESSING AND APPLICATIONS

LAB -1

GEOMETRIC TRANSFORMATION AND HISTOGRAM EQUILIZATION

STUDENTS NAME:

NAYEM AL TAREQ : 202293442

Kazi Salith Ur Rahman : 202291994

Procedure :

1. [1] Download the test image (im1.png) from Brightspace under Lab 01. Read the image, convert the image to a grayscale image, and display the image.


```
imc = imread('im1.png');% Read the image
img = rgb2gray(imc); % Convert to grayscale.
imshow(img); % View image
```

2. (a) Define the transformation matrix with a rotation angle of 45 degrees.

```
theta = 0.25*pi;
R = [cos(theta) sin(theta) 0; ...
-sin(theta) cos(theta) 0; ...
0 0 1];
```

(b) Calculate the size of the transformed image and generate an empty image with the calculated size.

```
[y_max, x_max] = size(img); % Obtaining the size of the
image
corners = [0,0,1; ...
x_max,0,1; ...
0,y_max,1; ...
x_max,y_max,1]; % Corner points of the image
new_corners = corners*R % New location of corners
new_width = round(max(new_corners(:,1)) - ...
min(new_corners(:,1)));
new_height = round(max(new_corners(:,2)) - ...
min(new_corners(:,2)));
new_img = zeros(new_height,new_width);
```

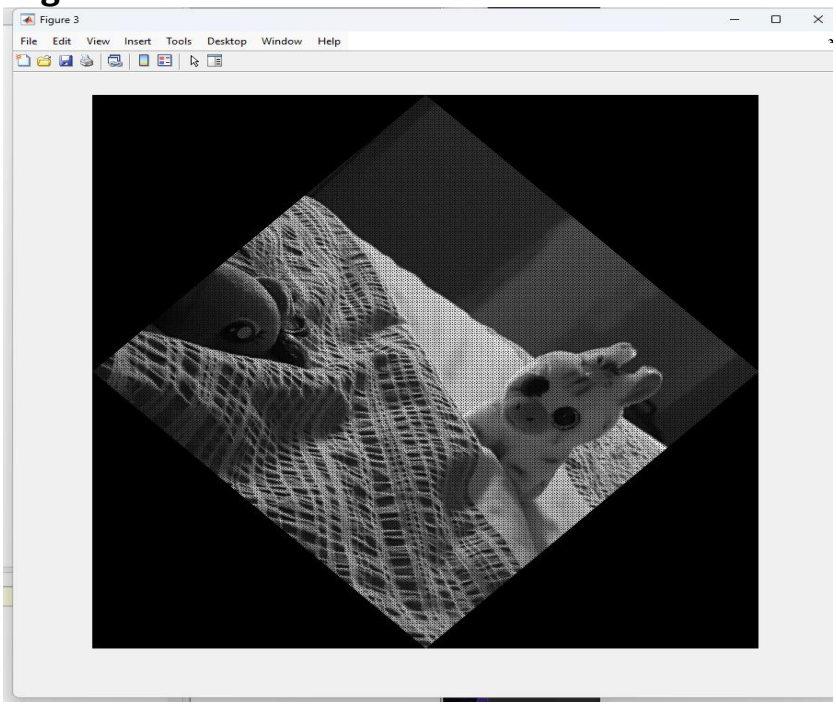
(c) Transform each pixel of the original image with the transformation matrix and assign the intensity to the new location.

```
min_width = abs(min(new_corners(:,1)));
min_height = abs(min(new_corners(:,2)));
min_width = round(min_width)+1;
min_height = round(min_height)+1;
rot_img = zeros(new_height,new_width);
  for i = 1:y_max
    for j = 1:x_max
     temp = ([j-1 i-1 1])*R;
     x_new = round(temp(1,1))+ min_width;
     y_new = round(temp(1,2))+ min_height;
     rot_img(y_new,x_new) = img(i,j);
   end
end
```

(d) [5] View the transformed image.

```
figure;
imshow(rot_img,[])
```

Result:45 Degree Rotation

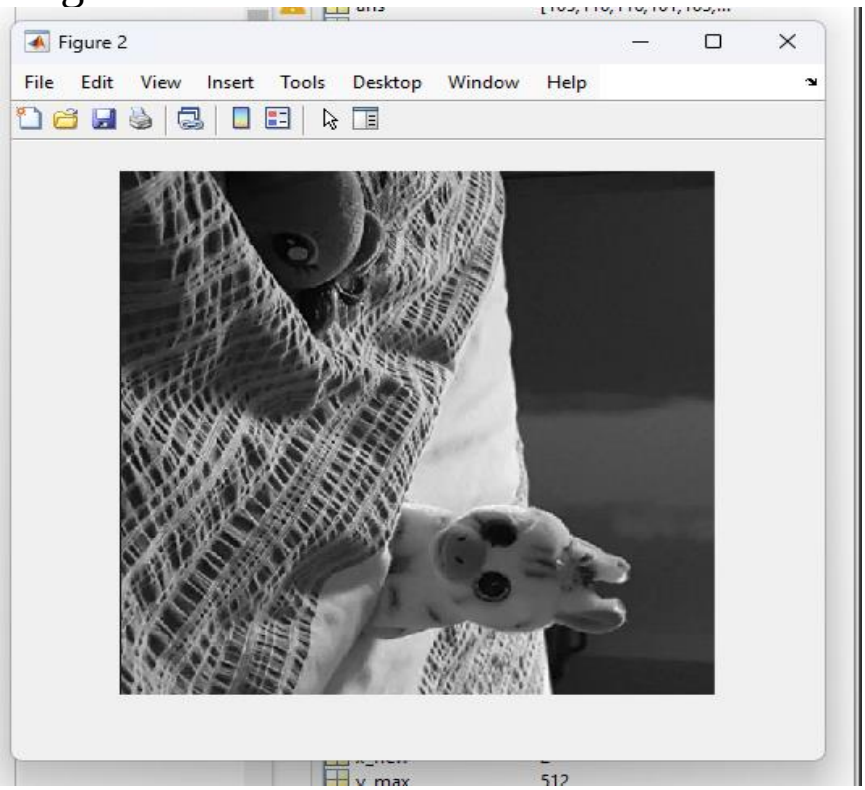


3. [5] Complete the steps above (Q2.a - 2.d) for angle θ = 90deg.

```
imc = imread('im1.png');% Read the image
img = rgb2gray(imc); % Convert to grayscale.
imshow(img); % View image


theta = 0.50*pi; %Rotation angle 90 degree
R = [cos(theta) sin(theta) 0; ...
-sin(theta) cos(theta) 0; ...
0 0 1];

[y_max, x_max] = size(img); % Obtaining the size of the image
corners = [0,0,1; ...
x_max,0,1; ...
0,y_max,1; ...
x_max,y_max,1]; % Corner points of the image
new_corners = corners*R % New location of corners
new_width = round(max(new_corners(:,1)) - ...
min(new_corners(:,1)));
new_height = round(max(new_corners(:,2)) - ...
min(new_corners(:,2)));
new_img = zeros(new_height,new_width);
min_width = abs(min(new_corners(:,1)));
min_height = abs(min(new_corners(:,2)));
min_width = round(min_width)+1;
min_height = round(min_height)+1;
```

```
rot_img = zeros(new_height,new_width);
    for i = 1:y_max
      for j = 1:x_max
       temp = ([j-1 i-1 1])*R;
       x_new = round(temp(1,1))+ min_width;
       y_new = round(temp(1,2))+ min_height;
       rot_img(y_new,x_new) = img(i,j);
      end
 end
 figure;
 imshow(rot_img,[])
```

Result: 90 deg Rotation

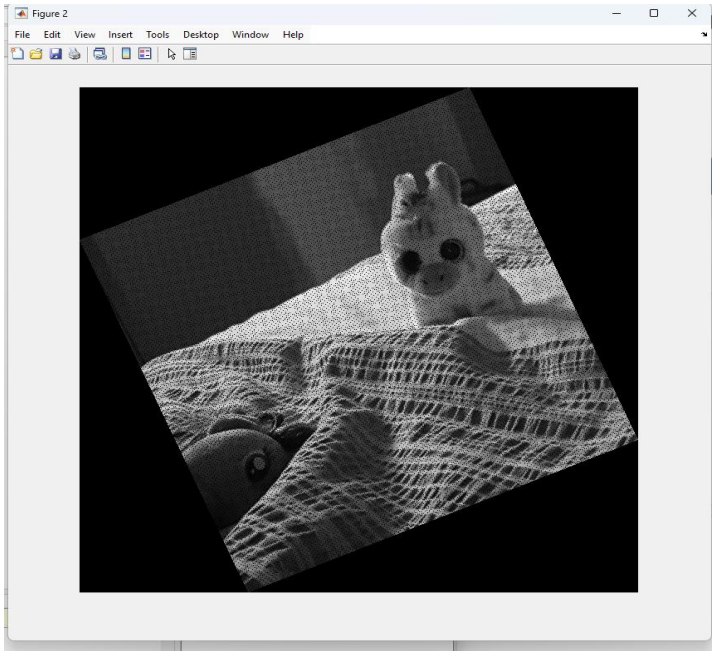


4. [5] Complete the steps above (Q2.a - 2.d) for angle θ = −25deg.

```
imc = imread('im1.png');% Read the image
img = rgb2gray(imc); % Convert to grayscale.
imshow(img); % View image


theta = -.13*pi; %Rotation angle -25 degree
R = [cos(theta) sin(theta) 0; ...
-sin(theta) cos(theta) 0; ...
0 0 1];
```

```
[y_max, x_max] = size(img); % Obtaining the size of the image
corners = [0,0,1; ...
x_max,0,1; ...
0,y_max,1; ...
x_max,y_max,1]; % Corner points of the image
new_corners = corners*R % New location of corners
new_width = round(max(new_corners(:,1)) - ...
min(new_corners(:,1)));
new_height = round(max(new_corners(:,2)) - ...
min(new_corners(:,2)));
new_img = zeros(new_height,new_width);
min_width = abs(min(new_corners(:,1)));
min_height = abs(min(new_corners(:,2)));
min_width = round(min_width)+1;
min_height = round(min_height)+1;
rot_img = zeros(new_height,new_width);
   for i = 1:y_max
      for j = 1:x_max
       temp = ([j-1 i-1 1])*R;
       x_new = round(temp(1,1))+ min_width;
       y_new = round(temp(1,2))+ min_height;
       rot_img(y_new,x_new) = img(i,j);
     end
end
figure;
imshow(rot_img,[])
```

Result: -25 Degree Rotation



5. [5] It can be seen that in some cases, when the image is transformed the output image has ‘empty’ black pixels. Explain why this happens in only some images, but not others.

It can be seen that in some cases, when the image is transformed the output image has black dots. In these cases, the image can be improved using the following method::

- Identify the boundary of the transformed image and create a blank image of equal dimensions.
- Use the inverse of the transformation matrix to map each pixel of the new blank image.
- For each point in the new image, assign the value of the closest pixel from the original image to the mapped location. If the mapped location falls outside the image, leave the value of the point as zero.

6. In the previous questions, we defined the size of the output image and mapped each input pixel to the corresponding output pixel. We can improve the output image, ensuring no ‘black’ or ‘empty’ pixels by first calculating the size of the transformed image, generating an empty image of the correct output size, and then calculate the inverse of the forward transformation matrix. For each pixel of the transformed output image, we use the inverse transformation to search backwards in the input image for the correct pixel. In this way, every pixel in the output image will find an input pixel. If the corresponding pixel in the input image is outside image boundary, we can simply set the value of the output pixel to zero.

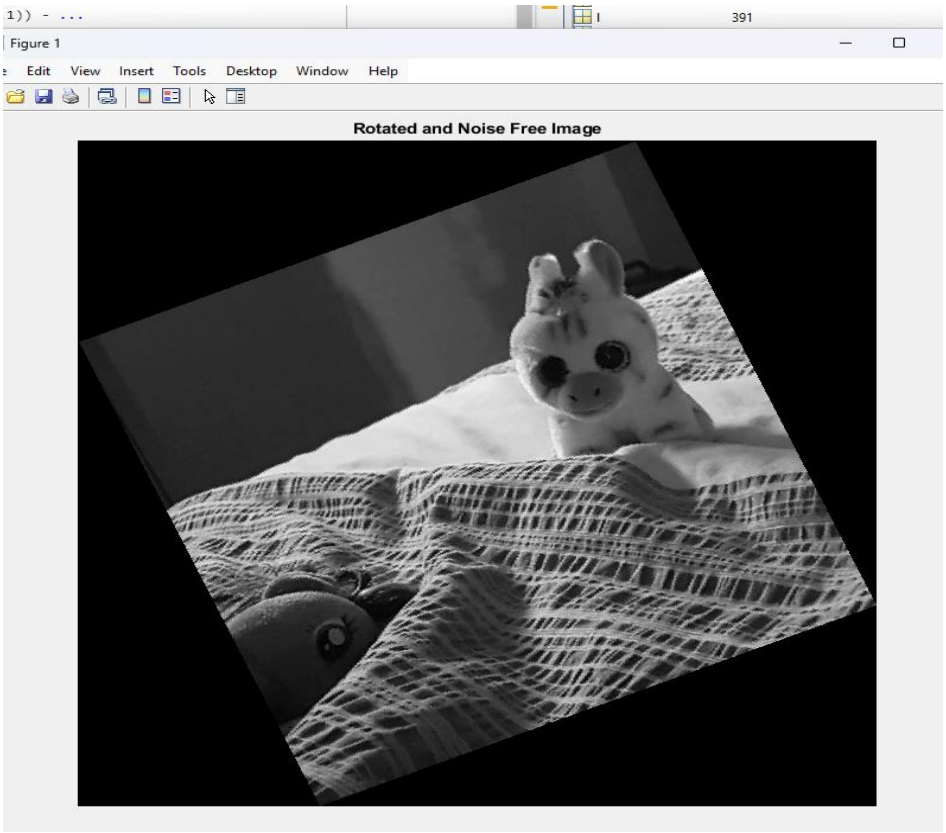
(a) Calculate the inverse transformation matrix.

(b) Modify the code in Q2 to calculate the following inverse transformation, and use it to iterate through each output image pixel to search the input image for the following transformations. Visualize your results.

```
imc = imread('im1.png');% Read the image
img = rgb2gray(imc); % Convert to grayscale.
imshow(img); % View image


theta = -.13*pi;
R = [cos(theta) sin(theta) 0; ...
-sin(theta) cos(theta) 0; ...
0 0 1];

[y_max, x_max] = size(img); % Obtaining the size of the image
corners = [0,0,1; ...
x_max,0,1; ...
0,y_max,1; ...
x_max,y_max,1]; % Corner points of the image
new_corners = corners*R % New location of corners
new_width = round(max(new_corners(:,1)) - ...
min(new_corners(:,1)));
new_height = round(max(new_corners(:,2)) - ...
```

```
min(new_corners(:,2)));
new_img = zeros(new_height,new_width);
min_width = abs(min(new_corners(:,1)));
min_height = abs(min(new_corners(:,2)));
min_width = round(min_width)+1;
min_height = round(min_height)+1;
rot_img = zeros(new_height,new_width);
       for i = 1:new_height
              for j = 1:new_width
              temp = ([j-min_width i-min_height 1])/R;
              x_new = round(temp(1,1));
              y_new = round(temp(1,2));
if (x_new>0 && x_new< x_max)&& (y_new > 0 && y_new < y_max)
    rot_img(i,j)=img(y_new,x_new);
else
    rot_img(i,j)=0;


  end
 end
end
figure,imshow(rot_img,[]);
title('Rotated and Noise Free Image');
```

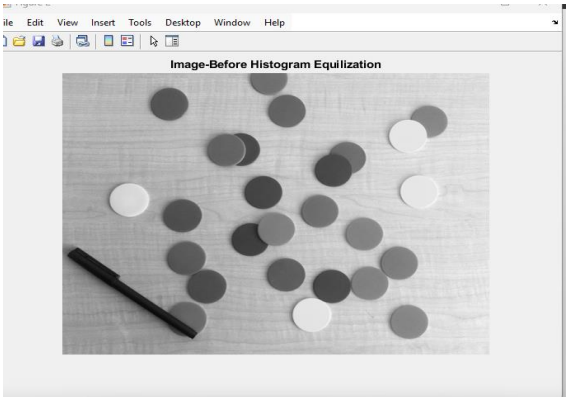
Result:



Histogram Equalization [24 points] :

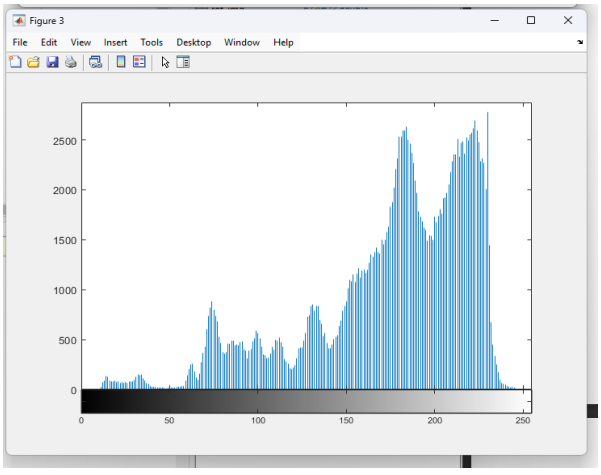
1. [1] Download the test image (im2.png) from Brightspace under Lab 01. Read the image, convert the image to a grayscale image, and visualize it.

```
imc = imread('im2.png');% Read the image
img = rgb2gray(imc); % Convert to grayscale
imshow(img); % View image
```



2. [5] Grayscale images have 256 gray levels. Therefore create an empty matrix having dimensions of 1 × 256, generate the histogram and display it

```
H=size(img,1); % Read the height of the image
W=size(img,2); % Read the width of the image
Hist_arr=zeros(1,256); % Array for holding the (original) histogram
Hist_eq_arr=zeros(1,256); % Array for holding the (equalized)histogram
CDF_array=zeros(1,length(Hist_arr)); % array to hold cumulative distribution
function (CDF)
hist_eq_img=uint8(zeros(H,W)); % A 2D array for keeping histogram equalized
image (intensity in 8 bit integers)
for i=1:H,
    for j=1:W,
    Hist_arr(1,img(i,j)+1)=Hist_arr (1,img(i,j)+1)+1 ;
    end
end
```



3. [5] Calculate pr(rk) for the image and display it. (Total number of pixels = ymax ×xmax).

```
Hist_arr_pdf=Hist_arr/(H*W); % PDF
```

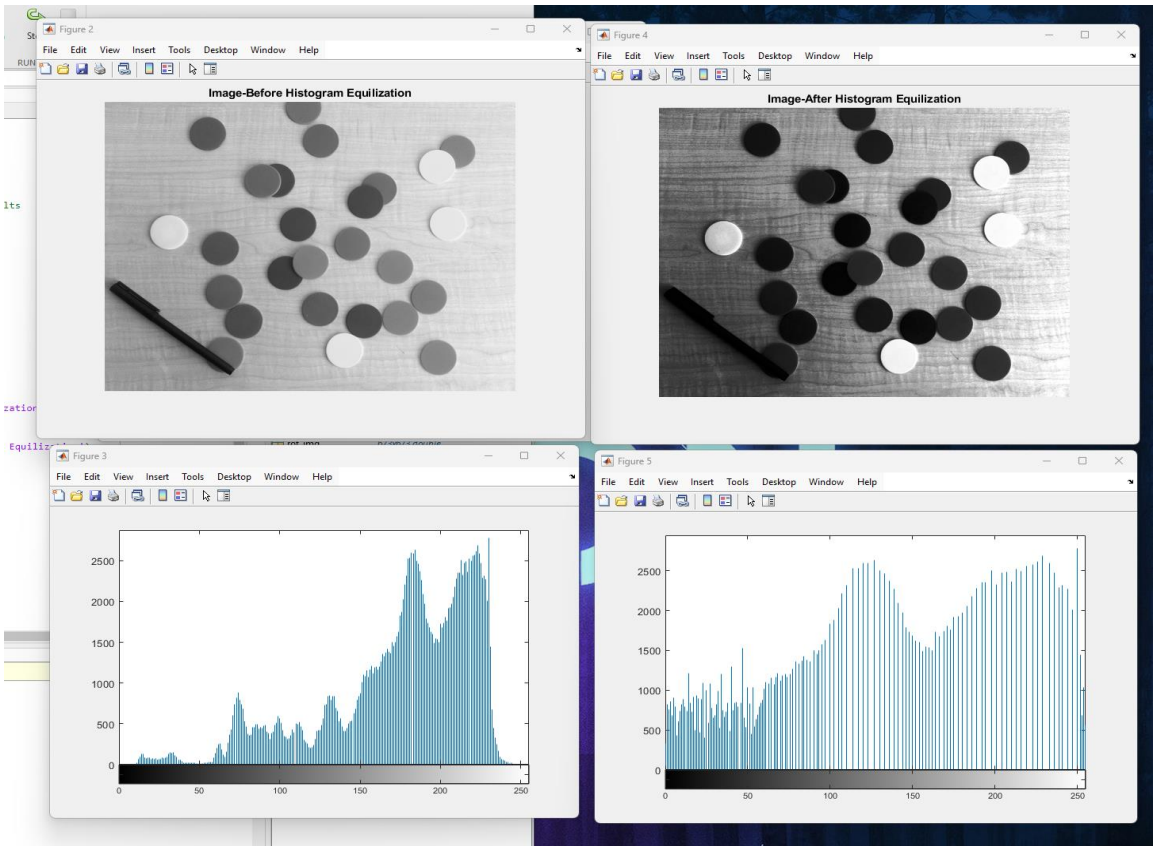
4. [5] Calculate the CDF from the PDF, and display it.

```
dummy1=0; % A dummy variable to hold the summation results
for k=1:length(Hist_arr); % Generating the CDF from PDF
dummy1=dummy1+Hist_arr_pdf(k);
CDF_array(k)= dummy1;
end
```

5. [3] Calculate the equalized histogram using equation 1 and map the original gray levels to the new gray levels. Display the original and new values

```
for l=1:H, % Histogram equalization
for m=1:W,
hist_eq_img(l,m)= round(CDF_array(img(l,m)+1)*...
(length(Hist_arr_pdf)-1)); % scale to 255 and round to
nearest integer
Hist_eq_arr(1, hist_eq_img(l,m)+1)=...
Hist_eq_arr (1,hist_eq_img(l,m)+1)+1 ; % Its histogram
end
end
```

6. [5] Display both images, before (original) and after histogram equalization.



Performance evaluation [10 points]

1. [5] Download or use any image that is larger than 640×640 pixels and use your code to rotate the image by 90 degrees. Use the inbuilt functions (e.g., maketform and imtransform in MATLAB Image Processing Toolbox) to perform the rotation of the same image. Comment on the difference in speed between the two approaches.

Solution :

The procedures are given below:

1. We have downloaded an image and saved it in the working directory.

2. We used Matlab for this task and wrote the below lines to read the image and convert the image to a grayscale image.

```
imc = imread('im1.png'); % Read the image
img = rgb2gray(imc); % Convert to grayscale. The supplied image is a grayscale, so
you don't need to use this step.
imshow(img); % View image
```

3. Defining the transformation matrix with a rotation angle of 90 degrees:

```
theta = 0.5*pi;
 R = [cos(theta) sin(theta) 0; ...
-sin(theta) cos(theta) 0; ...
0 0 1];
```

4. Calculating the size of the transformed image and generate an empty image with the calculated size:

```
 [y_max, x_max] = size(img); % Obtaining the size of the image
corners = [0,0,1; ...
x_max,0,1; ...
0,y_max,1; ...
x_max,y_max,1]; % Corner points of the image
new_corners = corners*R % New location of corners
new_width = round(max(new_corners(:,1)) - ...
min(new_corners(:,1)));
new_height = round(max(new_corners(:,2)) - ...
min(new_corners(:,2)));
new_img = zeros(new_height,new_width);
```

5. Transforming each pixel of the original image with the transformation matrix and assign the intensity to the new location:

```
min_width = abs(min(new_corners(:,1)));
min_height = abs(min(new_corners(:,2)));
min_width = round(min_width)+1;
min_height = round(min_height)+1;
rot_img = zeros(new_height,new_width);
for i = 1:y_max
    for j = 1:x_max
       temp = ([j-1 i-1 1])*R;
       x_new = round(temp(1,1))+ min_width;
       y_new = round(temp(1,2))+ min_height;
```

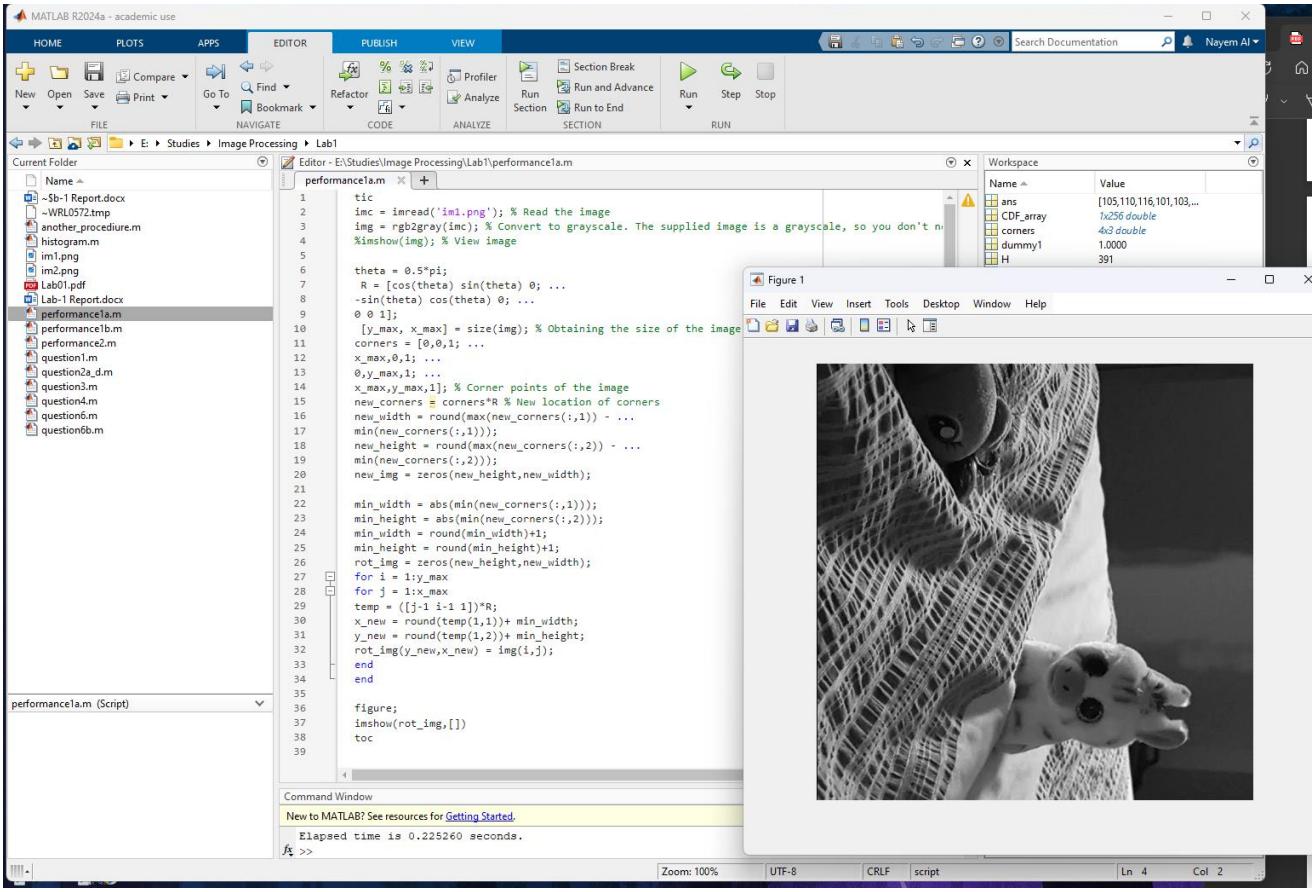
```
        rot_img(y_new,x_new) = img(i,j);
    end
end
```

6. Viewing the transformed image:

```
figure;
imshow(rot_img,[])
```

7. To measure the speed of the rotation for the image: We have used “tic” at the beginning of the code and “toc” at the end of the code. Thus, we get the output, which is given below:

```
toc
```



Here it can be seen that the speed for rotating the image by 90 degree is 5 samples:

T1: 0.225260 seconds

T2: 0.272229 seconds

T3: 0.247846 seconds

T4: 0.234401 seconds.

T5: 0.236915 seconds

Another Procedure:

We use the inbuilt functions (e.g., maketform and imtransform in MATLAB Image Processing Toolbox) to perform the rotation of the same image by 90 degrees.

1) We have used the same image which was saved in our working directory.

2) We used Matlab for this task and wrote the below lines to read the image and convert the image to a grayscale image.

```
imc = imread('im1.png'); % Read the image
img = rgb2gray(imc);
% Convert to grayscale. The supplied image is a grayscale, so you don't need to
use this step.
imshow(img); % View image
```

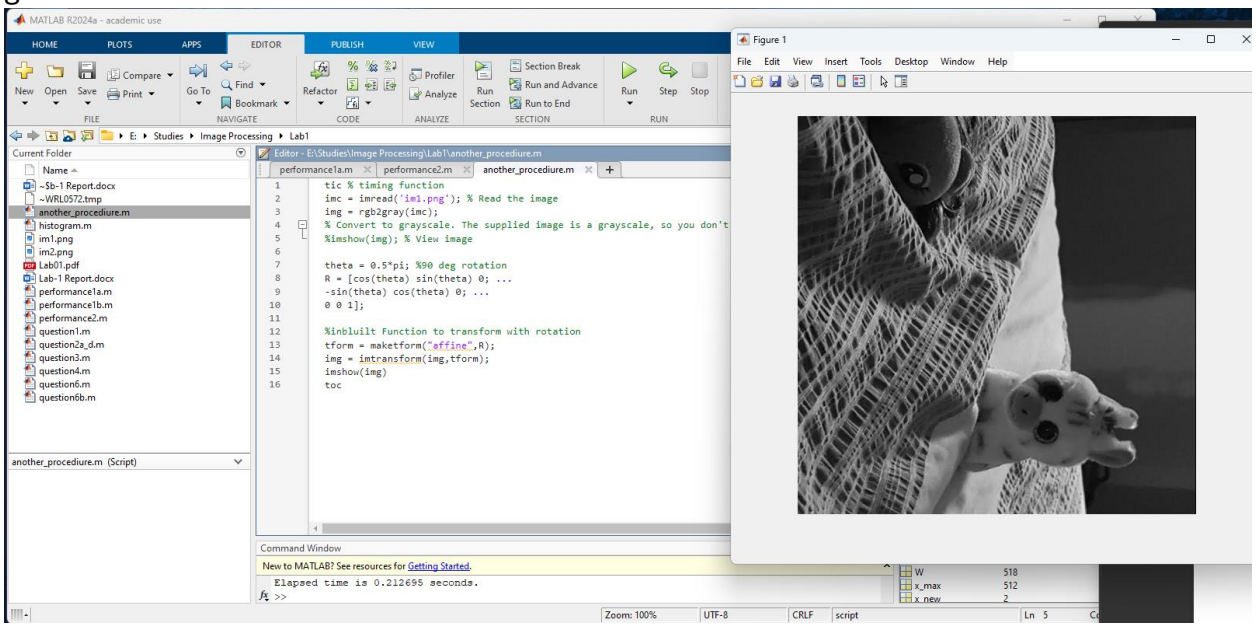
3) Defining the transformation matrix with a rotation angle of 90 degrees:

```
theta = 0.5*pi; %90 deg rotation
R = [cos(theta) sin(theta) 0; ...
-sin(theta) cos(theta) 0; ...
0 0 1];
```

4) Using the inbuilt functions to perform the rotation:

```
%inbluilt Function to transform with rotation
tform = maketform("affine",R);
img = imtransform(img,tform);
imshow(img)
```

5) To measure the speed of the rotation for the image: We have used “tic” at the beginning of the code and “toc” at the end of the code. Thus, we get the output, which is given below:



```
Elapsed time is in 5 samples:
T1: 0.212695 seconds.
T2: 0.078012 seconds.
T3: 0.065429 seconds.
T4:0.064797 seconds.
T5: 0.066908 seconds.

In the conclusion , it can be said that codes with inbuilt function takes less
time to run because less codes to process and read.
```

2. [5] Comment on the shape of the histogram resulting from the histogram equalization process. Is it a perfectly uniform histogram, like we would theoretically expect? Provide an explanation of why it is, or is not, uniform.

The procedure is given below:
1. We downloaded the test image (im2.png) from Brightspace under Lab 01 and saved it in the working directory.
2. Reading the image and converting the image to a grayscale image.

```
imc = imread('im2.png');% Read the image
img = rgb2gray(imc); % Convert to grayscale
imshow(img); % View image
```

3. Grayscale images have 256 gray levels. Therefore creating an empty matrix having dimensions of 1 × 256 and generate the histogram.

```
%Grayscale images have 256 gray levels
H=size(img,1); % Read the height of the image
W=size(img,2); % Read the width of the image
Hist_arr=zeros(1,256); % Array for holding the (original) histogram
Hist_eq_arr=zeros(1,256); % Array for holding the (equalized) histogram
CDF_array=zeros(1,length(Hist_arr)); % array to hold cumulative distribution
function
%(CDF)
hist_eq_img=uint8(zeros(H,W)); % A 2D array for keeping histogram equalized image
%(intensity in 8 bit integers)
for i=1:H,
for j=1:W,
Hist_arr(1,img(i,j)+1)=Hist_arr (1,img(i,j)+1)+1 ;
end
end
```

4. Calculating pr(rk) for the image. (Total number of pixels = ymax × xmax)

```
%Calculating pr(rk) for the image. (Total number of pixels = ymax × xmax)
Hist_arr_pdf=Hist_arr/(H*W); % PDF
```

5. Calculating the CDF from the PDF

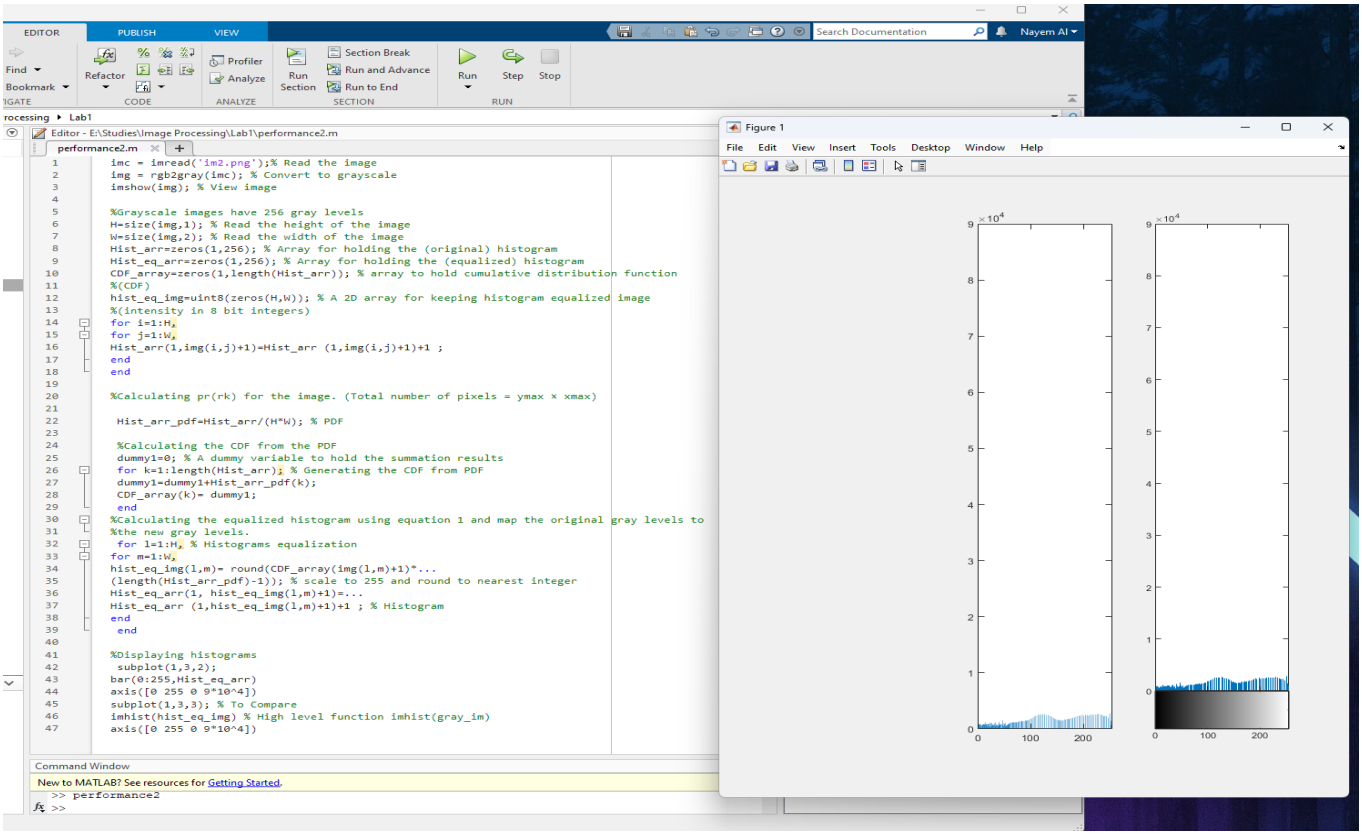
```
%Calculating the CDF from the PDF

    dummy1=0; % A dummy variable to hold the summation results
    for k=1:length(Hist_arr); % Generating the CDF from PDF
    dummy1=dummy1+Hist_arr_pdf(k);
    CDF_array(k)= dummy1;
    end
```

6. Calculating the equalized histogram using equation 1 and map the original gray levels to the new gray levels.

```
%Calculating the equalized histogram using equation 1 and map the original gray
levels to
%the new gray levels.
 for l=1:H, % Histograms equalization
for m=1:W,
hist_eq_img(l,m)= round(CDF_array(img(l,m)+1)*...
(length(Hist_arr_pdf)-1)); % scale to 255 and round to nearest integer
Hist_eq_arr(1, hist_eq_img(l,m)+1)=...
Hist_eq_arr (1,hist_eq_img(l,m)+1)+1 ; % Histogram
end
 end
```

7. Displaying histograms

```
%Displaying histograms
 subplot(1,3,2);
bar(0:255,Hist_eq_arr)
axis([0 255 0 9*10^4])
subplot(1,3,3); % To Compare
imhist(hist_eq_img) % High level function imhist(gray_im)
axis([0 255 0 9*10^4])
```



Comments:

From the image of Matlab code and output histogram , we can say that it is not a perfect uniform histogram as the value are not repeating thus syntax of the graph is unique.