EECE-5626 (IP&PR) : Homework-6

Due on November 19, 2021 by 11:59 pm via submission portal

NAME: McKean, Tyler

Instructions

1. You are required to complete this assignment using Live Editor.
2. Enter your MATLAB script in the spaces provided. If it contains a plot, the plot will be displayed after the script.
3. All your plots must be properly labeled and should have appropriate titles to get full credit.
4. Use the equation editor to typeset mathematical material such as variables, equations, etc.
5. After completeing this assignment, export this Live script to PDF and submit the PDF file through the provided submission portal.
6. You will have two attempts to submit your assignment. However, make every effort to submit the correct and completed PDF file the first time. If you use the second attempt, then that submission will be graded.
7. Your submission of problem solutions must be in the given order, i.e., P1, P2, P3, etc. Do not submit in a random order.
8. Please submit your homework before the due date/time. A late submission after midnight of the due date will result in loss of points at a rate of 10% per hour until 8 am the following day, at which time the solutions will be published.

Reading Assignment:

- Chapter 10 from DIP4E, and
- Chapter 11 from DIPUM3E.

Default Plot Parameters:

```
set(0,'defaultfigurepaperunits','points','defaultfigureunits','points');
set(0,'defaultaxesfontsize',10);
set(0,'defaultaxestitlefontsize',1.4,'defaultaxeslabelfontsize',1.2);
```

Table of Contents

Problem-6.1: DIP4E Problem 10.14 (Page 868)

Solution:

```
f = imread('fingerprint.tif');
g = ones(3);
sobel = [-1, -2, -1; 0, 0, 0; 1, 2, 1]';
f_filt = conv2(f,sobel);
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1), imshow(f), title('Original Image')
subplot(1,2,2), imshow(f_filt), title('Sobel Filtered Image')
```


```
diff = [-1, 0, 1]';
smooth = [1, 2, 1];
array = conv2(diff,smooth)
```

```
array = 3×3
    -1    -2    -1
```

```
     0     0     0
     1     2     1
```

```
f_diff = conv2(f,diff);
f_smooth = conv2(f_diff,smooth);
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1), imshow(f_diff), title('Difference Kernel Result')
subplot(1,2,2), imshow(f_smooth), title('Smoothing Kernel Result')
```


```
equal = (sum(f_filt(:)) == sum(f_smooth(:)))
```

```
equal = logical
   1
```

The result of passing the 1D difference kernel [-1 0 1] followed up with the 1D smoothing kernel [1 2 1] results in the equivalent operation as filtering the image with a Sobel filter kernel.

Problem-6.2: DIP4E Problem 10.23 (Page 868)

Do the following.

(a) Marr-Hildreth Algorithm

Show that Steps 1 and 2 of the Marr-Hildreth (MH) algorithm can be implemented using four 1-D convolutions.

Solution:

```
f = ones(3);
laplac = [1, -2, 1];
Gx = [1, 2, 1];
Gy = [1; 2; 1];
G = conv2(Gx,Gy)/16
```

```
G = 3×3
    0.0625    0.1250    0.0625
    0.1250    0.2500    0.1250
    0.0625    0.1250    0.0625
```

```
Gf = conv2(G,f)
```

```
Gf = 5×5
```

```
    0.0625    0.1875    0.2500    0.1875    0.0625
    0.1875    0.5625    0.7500    0.5625    0.1875
    0.2500    0.7500    1.0000    0.7500    0.2500
    0.1875    0.5625    0.7500    0.5625    0.1875
    0.0625    0.1875    0.2500    0.1875    0.0625
```

```
gx = imfilter(Gf,laplac)
```

```
gx = 5×5
    0.0625   -0.0625   -0.1250   -0.0625    0.0625
    0.1875   -0.1875   -0.3750   -0.1875    0.1875
    0.2500   -0.2500   -0.5000   -0.2500    0.2500
    0.1875   -0.1875   -0.3750   -0.1875    0.1875
    0.0625   -0.0625   -0.1250   -0.0625    0.0625
```

```
gy = imfilter(Gf,laplac')
```

```
gy = 5×5
    0.0625    0.1875    0.2500    0.1875    0.0625
   -0.0625   -0.1875   -0.2500   -0.1875   -0.0625
   -0.1250   -0.3750   -0.5000   -0.3750   -0.1250
   -0.0625   -0.1875   -0.2500   -0.1875   -0.0625
    0.0625    0.1875    0.2500    0.1875    0.0625
```

```
LoG = gx + gy
```

```
LoG = 5×5
    0.1250    0.1250    0.1250    0.1250    0.1250
    0.1250   -0.3750   -0.6250   -0.3750    0.1250
    0.1250   -0.6250   -1.0000   -0.6250    0.1250
    0.1250   -0.3750   -0.6250   -0.3750    0.1250
    0.1250    0.1250    0.1250    0.1250    0.1250
```

Let image be $f(x, y)$ and the Gaussian operator be $G(x, y)$ and the Laplacian operator be ∇^2.

The Marr-Hildreth algorithm can be expressed as:

$$g(x, y) = \nabla^2\{G(x, y) \star f(x, y)\} = \frac{\partial}{\partial x^2}\{G(x, y) \star f(x, y)\} + \frac{\partial}{\partial y^2}\{G(x, y) \star f(x, y)\}$$

The 2D convolutions can be broken down into two 1D convolutions, such that

$$G(x, y) \star f(x, y) = G(x) \star G(y) \star f(x, y)$$

Which results in the Marr-Hildreth algorithm being

$$g(x, y) = \nabla^2\{G(x) \star G(y) \star f(x, y)\} = \frac{\partial}{\partial x^2}\{G(x) \star G(y) \star f(x, y)\} + \frac{\partial}{\partial y^2}\{G(x) \star G(y) \star f(x, y)\}$$

Thus, the MH algorithm can be represented by four 1D convolutions

(b) Derivation of Expression

Show that the expression for the computational advantage of using the 1-D convolution approach as in (a) as opposed to implementing the 2-D convolution directly is

$$A = \frac{n^2 + 9}{2n + 6}.$$

See the rest of the problem statement about more details on discretization. Also, study Problem 10.22.

Solution:

The 2D convolutions of the Marr-Hildreth algorithm require $n \times n$ multiplations for all $M \times N$ pixels of $G(x, y) \star f(x, y)$ so, $n^2 \times M \times N$

The convolution of the 3×3 Laplacian mask is then required which factors in $9 \times M \times N$ multiplications, Which gives the total mulitplications for the 2D convolutions to be $n^2 \times M \times N + 9 \times M \times N \Rightarrow (n^2 + 9)MN$

The four 1D convolutions of part(a) would result in $n \times M \times N$ mulitplications for two convolutions and another $3 \times M \times N$ multiplications for the 1×3 Laplacian Mask.

This results in $2(n \times M \times N) + 2(3 \times M \times N) \Rightarrow 2MN(n + 3)$

Thus the Computational advantage of 1D convolutions with respect to the 2D convolutions would be

$$\frac{(n^2 + 9)MN}{2MN(n + 3)} = \frac{n^2 + 9}{2n + 6}$$

Problem-6.3: Sobel, Prewitt, and Kirsch Compass Kernels

(a) DIP4E Project 10.2(a) (Page 874)

Note that the name of the function is changed to edgeKernel6H in the following function code (6H for the 6th homework).

MATLAB function: Enter your code for **`edgeKernel6H`** below after comments and create your function at the end of this file.

```
function w = edgeKernel6H(type,dir)
% EDGEKERNEL6H spatia1 edge kerne1.
% G = EDGEKERNEL6H(TYPE,DIR) generates a 3-by—3 edge kernel of
% specified TYPE and direction (DIR). These kernels are shown in
% Figs. 10.14 and 10.15 of DIP4E.
%
% Valid TYPE/DIR pairs are:
% TYPE DIR
% 'prewitt' 'v'/'h'
% 'sobe1' 'v'/'h‘
% 'kirsch' 'n'/'nw'/'w'/'sw'/'s'/'se'/'e'/'ne'

% Enter your code below
switch type
    case 'prewitt'
        switch dir
            case 'v'
                w = [-1,0,1;-1,0,1;-1,0,1];
            case 'h'
                w = [-1,-1,-1;0,0,0;1,1,1];
        end
    case 'sobel'
        switch dir
            case 'v'
```

```
                w = [-1,0,1;-2,0,2;-1,0,1];
            case 'h'
                w = [-1,-2,-1;0,0,0;1,2,1];
        end
    case 'kirsch'
        switch dir
            case 'n'
                w = [-3,-3,5;-3,0,5;-3,-3,5];
            case 'nw'
                w = [-3,5,5;-3,0,5;-3,-3,-3];
            case 'w'
                w = [5,5,5;-3,0,-3;-3,-3,-3];
            case 'sw'
                w = [5,5,-3;5,0,-3;-3,-3,-3];
            case 's'
                w = [5,-3,-3;5,0,-3;5,-3,-3];
            case 'se'
                w = [-3,-3,-3;5,0,-3;5,5,-3];
            case 'e'
                w = [-3,-3,-3;-3,0,-3;5,5,5];
            case 'ne'
                w = [-3,-3,-3;-3,0,5;-3,5,5];
        end
end
```

(b) Kirsch Kernels

Using your **edgeKernel6H** function, generate north-west and south oriented Kirsch kernels and display them as matrices. See solutions to Project 10.2(b) for this display. DO NOT use the **edgeKernel4e** function from Student Resources. You can use it to verify your reults.

MATLAB script:

```
clc; close all; clear;
NWkirsch = edgeKernel6H('kirsch','nw')
```

```
NWkirsch = 3×3
    -3     5     5
    -3     0     5
    -3    -3    -3
```

```
Skirsch = edgeKernel6H('kirsch','s')
```

```
Skirsch = 3×3
     5    -3    -3
     5     0    -3
     5    -3    -3
```

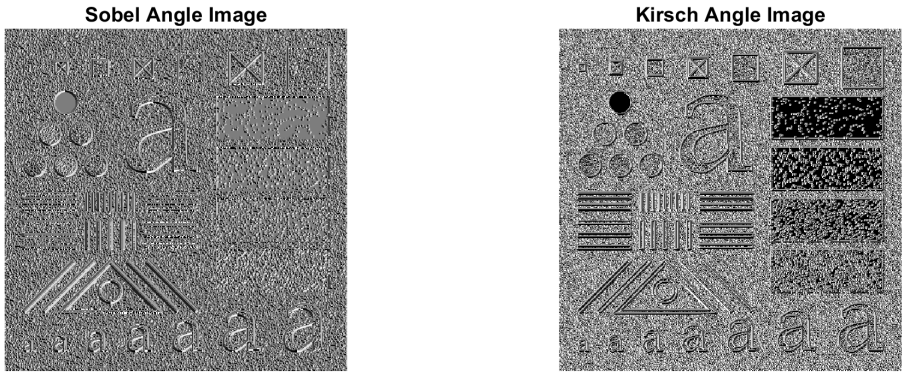
(c) DIP4E Project 10.4(b) (Page 875)

For this part use the **edgeAngle4e** function from your DIP4E Student Resources.

MATLAB script:

```
f = imread('testpattern512.tif');
gs = edgeAngle4e(f,'sobel');
gk = edgeAngle4e(f,'kirsch');
figure('Units','inches','Position',[0,0,12,4]);
```

```
subplot(1,2,1); imshow(intScaling4e(gs,'full')); title('Sobel Angle Image','fontsize',14);
subplot(1,2,2); imshow(intScaling4e(gk,'full')); title('Kirsch Angle Image','fontsize',14);paus
```



Comment:

The angle image of the Kirsch filter has more of an edge resolution than the Sobel angle image, because the Kirsch filter is designed to detect edge magnitude and direction in all eight compass directions. Whereas, the sobel filter can only account for vertical and horizontal directions, so the Kirsch filter has better edge detections for diagonal $+45^{\circ}$ and -45° directions

(b) DIP4E Project 10.4(c) (Page 875)

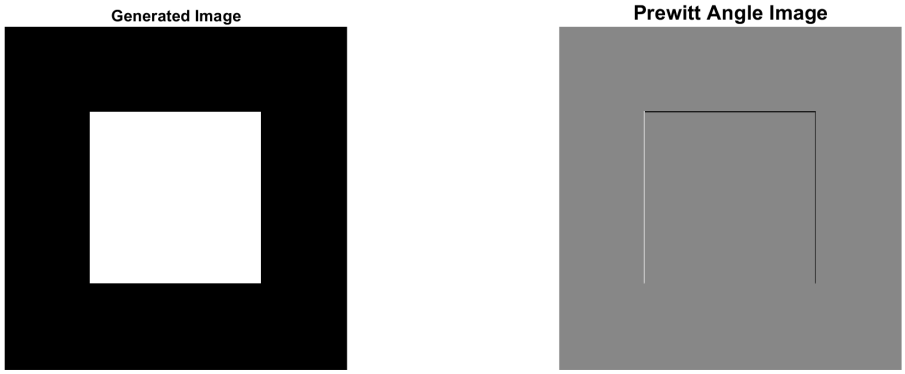
For this part also, use the **edgeAngle4e** function from your DIP4E Student Resources.

MATLAB script:

```
M = 512; N = 512;
image = zeros(M,N);
image(M/4:(3*M/4)-1,N/4:(3*N/4)-1) = 1;
sum = sum(sum(image)) % 256 x 256 center should be 65536 white pixels
```

```
sum = 65536
```

```
gp = edgeAngle4e(image,'prewitt');
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(image), title('Generated Image')
subplot(1,2,2); imshow(intScaling4e(gp,'full')); title('Prewitt Angle Image','fontsize',14);pau
```



```
neg = gp < 0;
pos = gp > 0;
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(neg), title('Negative Values')
subplot(1,2,2); imshow(pos); title('Positive Values');pause(1)
```

Comments:

The prewitt kernel that was implemented from the edgeAngle4e() function was a horizontal prewitt kernel. Thus, when computing the edges of the internal square of the generated image, we get one vertical line of positive angle values, and two lines of negative angle values. The bottom edge of the square was computed as a 0° angle, which is why no edge appears. The positive values are a $+90^{\circ}$ angle, and then the two negative values are $180^{\circ}\text{and}\,270^{\circ}$ or $-90^{\circ} and -180^{\circ}$ angles.

Problem-6.4: DIP4E Project 10.6 (Page 875)

Basic Global Thresholding

(a) Function `globalThresh6H`

Note that the name of the function is changed to **`globalThresh6H`** in the following function code (6H for the 6th homework).

MATLAB function: Enter your code below after comments and create your function at the end of this file.

```
function [g,T,k] = globalThresh6H(f,delT)
% GLOBALTHRESH6H Simple global thresholding.
% [G,T,K] = GLOBALTHRESH6H(F,DELT) iteratively computes a global
% threshold, T, and uses it to segment image F into two regions
% with values 0 for pixels <= T and 1 otherwise. The segmented
% image is output as G. The algorithm iterates until the difference
% between successive values of T are <= DELT. The final number of
% such iterations is output as K. The program uses the average
% intensity of F as the initial threshold. The value of DELT must
% be a nonnegative integer; it defaults to DELT = 0.01. The
% program normalizes the intensity values of F to the range [0,1].

% Enter your code below
```

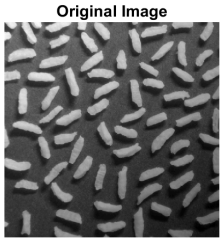
```
if nargin < 2
    delT = 0.01;
end
if delT < 0
    error('delT must be a nonnegative integer!');
end
k = 0;
f = intensityScaling(f); % scale intensity values of f to range [0,1]
T = mean2(f);
k = 0;
done = false;
while ~done
    k = k + 1;
    g = f > T;
    newT = 0.5*(mean(f(g)) + mean(f(~g)));
    done = abs(T - newT) < delT;
    T = newT;
end
end
```

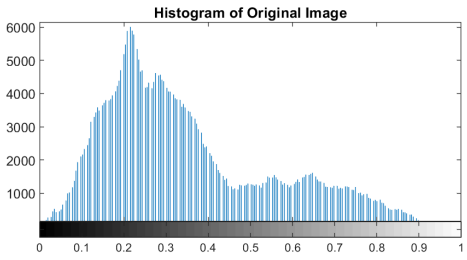
(b) Thresholding of `rice-shaded.tif` Image

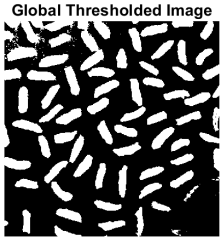
For this part use your **`globalThresh6H`** function. DO NOT use the **`globalThresh6H`** function from Student Resources. You can use it to verify your reults.

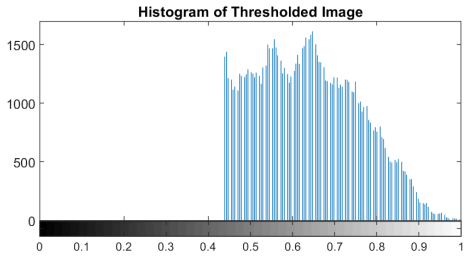
MATLAB script:

```
clc; clear; close all;
f = intensityScaling(imread('rice-shaded.tif'));
[g,T,k] = globalThresh6H(f);
figure('Units','inches','Position',[0,0,12,6]);
subplot(2,2,1); imshow(f), title('Original Image')
subplot(2,2,2); imhist(intensityScaling(f)); title('Histogram of Original Image');xlim([0.000 1
subplot(2,2,3); imshow(g), title('Global Thresholded Image')
subplot(2,2,4); imhist(f(g)); title('Histogram of Thresholded Image');pause(1)
```









```
display(k)
```

```
k = 4
```

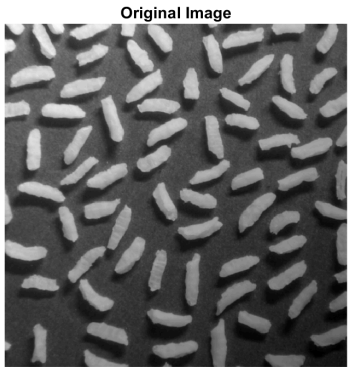
```
display(T)
```

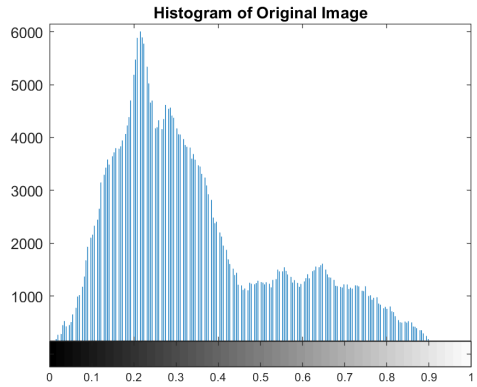
```
T = 0.4414
```

(c) Is there a Better Value of Parameter delT?

MATLAB script:

```
f = intensityScaling(imread('rice-shaded.tif'));
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(f), title('Original Image')
subplot(1,2,2); imhist(intensityScaling(f)); title('Histogram of Original Image');xlim([0.000 1
```





In terms of segmentation, the thresholding algorithm will only work well in situations where there is a reasonably clear valley between the modes of the histogram. Looking at the histogram of the original image above, there is no clear valley between the average pixel intensity value of T = 0.4414. This means that there is no better value for the parameter ΔT.

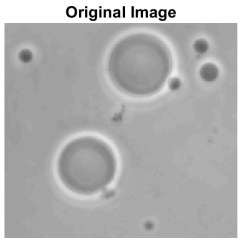
Problem-6.5: Otsu Global Thresholding

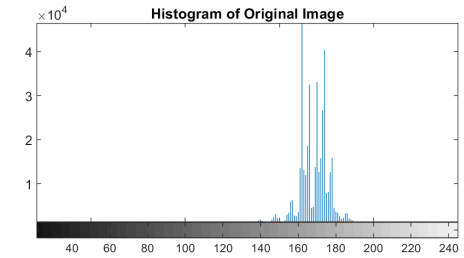
For this problem use the **`otsuThresh4e`** function from Student Resources.

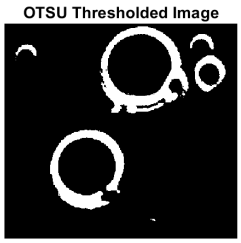
(a) DIP4E Project 10.7(b) (Page 875)

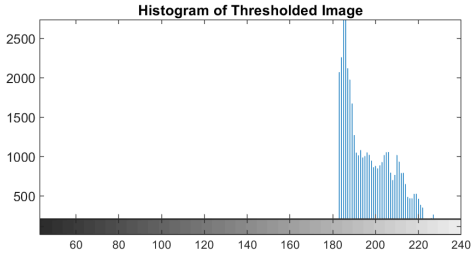
MATLAB script:

```
clc; clear; close all;
f = imread('polymercell.tif');
[g,~,thresh] = otsuThresh4e(f);
figure('Units','inches','Position',[0,0,12,6]);
subplot(2,2,1); imshow(f), title('Original Image')
subplot(2,2,2); imhist(f); title('Histogram of Original Image'), xlim([21 245]),ylim([1303 4649
subplot(2,2,3); imshow(g), title('OTSU Thresholded Image')
subplot(2,2,4); imhist(f(g)); title('Histogram of Thresholded Image'), xlim([43 240]), ylim([21
```









Confirmation:

From Example 10.14 in the DIP4E book, the threshold value computed by Otsu's method was 182, and the value computed using the otsuThresh4e() functions was:

```
display(thresh)
```

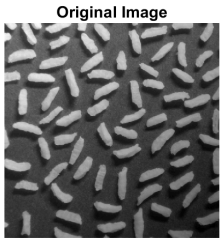
```
thresh = 182
```

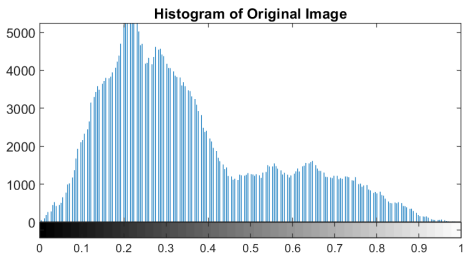
which confirms the results are the same.

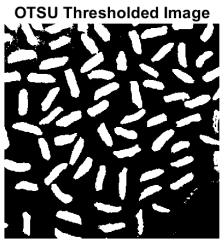
(b) DIP4E Project 10.7(c) (Page 875)

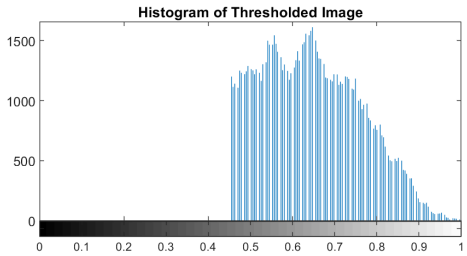
MATLAB script:

```
f1 = intensityScaling(imread('rice-shaded.tif'));
[g1,sep,thresh1] = otsuThresh4e(f1);
figure('Units','inches','Position',[0,0,12,6]);
subplot(2,2,1); imshow(f1), title('Original Image')
subplot(2,2,2); imhist(f1); title('Histogram of Original Image');pause(1)
subplot(2,2,3); imshow(g1), title('OTSU Thresholded Image')
subplot(2,2,4); imhist(f1(g1)); title('Histogram of Thresholded Image');pause(1)
```









Comments:

The threshold value for the basic global thresholding algorithm was about 0.4414, whereas the result of the OTSU algorithm outputted a threshold value of 0.4471. These relate to the mean pixel intensity value of about 113-114, when rounded to the nearest integer. Given the nature of the histogram, the separability between pixel values was still too low to cause any significant improvement over the basic algorithm. Again, this is because there isn't a distinct presence of a valley between the two modes, thus both thresholding algorithms resulted in an identical thresholding value as a result.

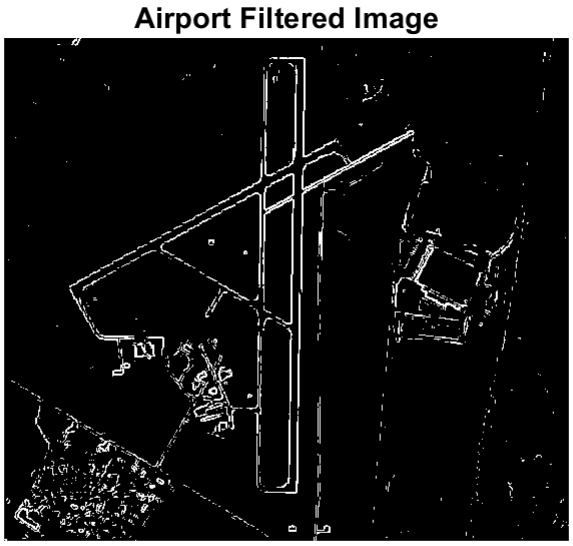
Problem-6.6: Hough Transform

Read the image **`airport.tif`** and obtain an edge map suitable for the following parts. Also, you will need to use the **`hough`**, **`houghpeaks`**, and **`houghlines`** functions from the IPT.

(a) DIPUM3E MATLAB Project 11.4 (b) (Page-718).

MATLAB script:

```
clc; close all; clear;
f = im2double(imread('airport.tif'));
g = edge(f,'sobel',0.13,'nothinning');
figure, imshow(g), title('Airport Filtered Image')
```

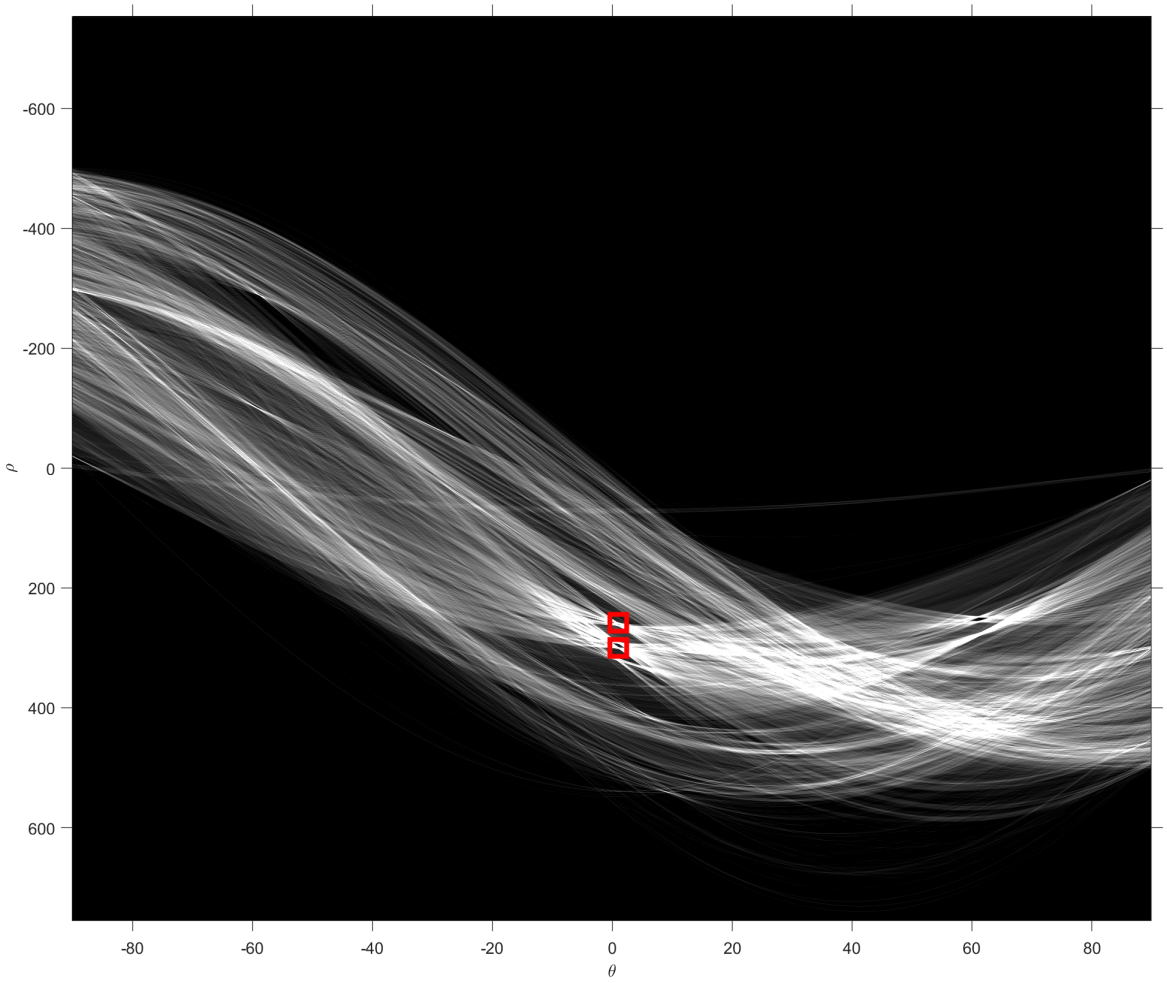



```
[H, thetaV, rhoV] = hough(g,'thetaRes',0.1);
figure('Units','inches','Position',[0,0,12,4]), imshow(imadjust(mat2gray(H)), 'XData',thetaV,'Y
daspect auto, axis on, xlabel('\theta'), ylabel('\rho')
```

(b) DIPUM3E MATLAB Project 11.4 (c) (Page-718).

MATLAB script:

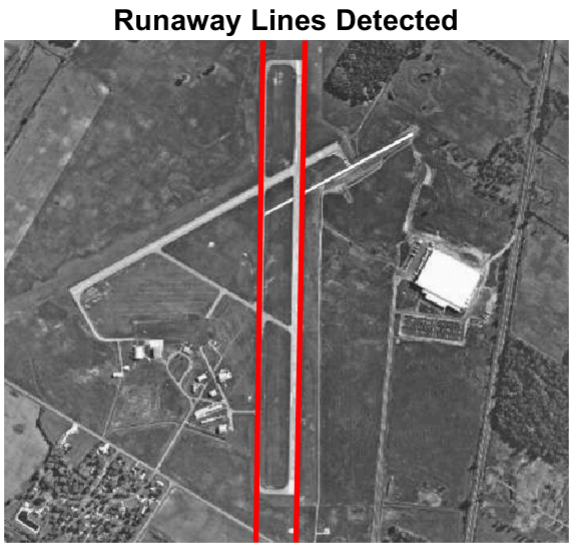
```
peaks = houghpeaks(H,2);
hold on
plot(round(thetaV(peaks(:,2))),rhoV(peaks(:,1)),'LineStyle',...
    'none','LineWidth',3,'marker','s','MarkerSize',14,'color','r')
hold off
```



(c) DIPUM3E MATLAB Project 11.4 (d) (Page-718).

MATLAB script:

```
lines = houghlines(f,thetaV,rhoV,peaks,'FillGap',1.5);
figure, imshow(f); hold on
for i = 1:length(lines)
    xy = [lines(i).point1; lines(i).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','r')
end
hold off
title('Runaway Lines Detected ')
```



Problem-6.7: Moving Average Thresholding

Do the following.

(a) DIPUM3E MATLAB Project 11.6 (d) (Page-718).

MATLAB function: Enter your code below after comments and create your function at the end of this file.

```
function g = movingaveragethresh(f,n,K)
% MOVINGAVERAGETHRESH Image segmentation using a moving average threshold.
% G = MOVINGAVERAGETHRESH(F,n,K) segments image F by thresholding its
% intensities based on the moving average of the intensities along
% individua1 rows of the image. The average at pixel k is formed by
% averaging the intensities of that pixel and its n – 1 preceding
% neighbors using Eq. (11-23). To reduce shading bias, the scanning is
% done in a zig–zag manner, treating the pixe1s as if they were a 1-D,
% continuous stream. If the value of the image at a point exceeds K
% percent of the value of the running average at that point, a 1 is
% output in that location in G. otherwise a 0 is output. At the end of
% the procedure, G is thus the thresho1ded (segmented) image. K is a
% sca1ar in the range [0,1].

% Enter your code below
if (K > 1 || K < 0)
    error('K must be a scalar within range [0,1]')
end
[M,N] = size(f);
m = zeros(1,N);
g = false(M,N);
window = n;
c = (1/window)*ones(1,window);
d = 1;
for k = 1:M
```

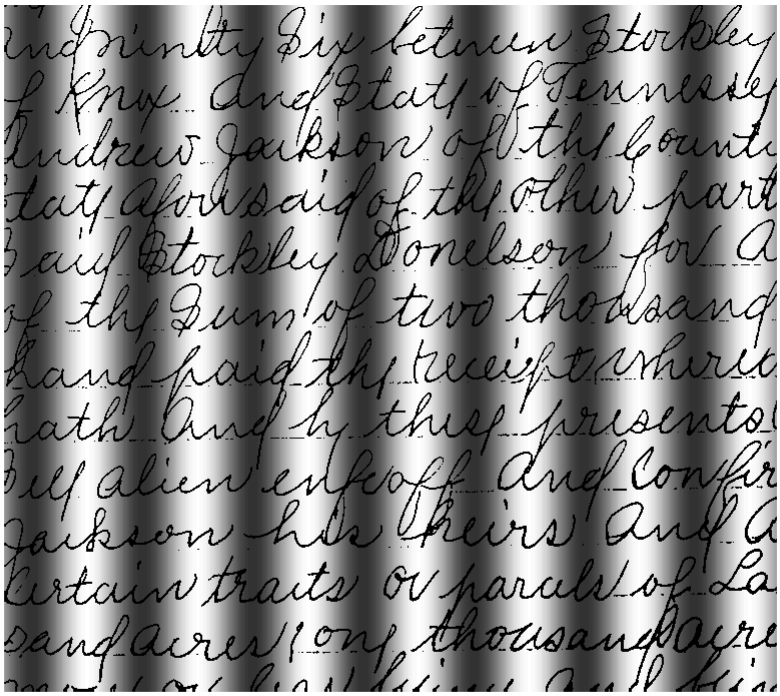
```
        m(k,:) = filter(c,d,f(k,:));
    end
    for i = 1:M
        for j = 1:N
            if f(i,j) > K*m(i,j)
                g(i,j) = true;
            else
                g(i,j) = false;
            end
        end
    end
    end
```

(b) DIPUM3E MATLAB Project 11.6 (f) (Page-718).

Segment the image **`text-sineshade.tif`**.

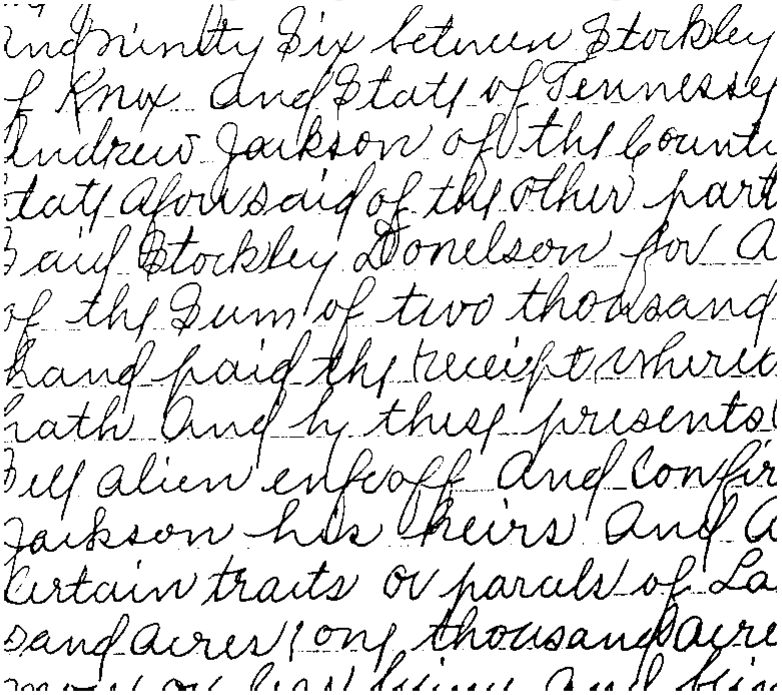
MATLAB script: Insert your code below.

```
clc; close all; clear;
f = imread('text-sineshade.tif');
figure, imshow(f)
```



```
g = movingaveragethresh(f,20,0.5);
figure, imshow(g), title('Moving Average Thresholded Image')
```

Moving Average Thresholded Image



Problem-6.8 Thresholding of Compound Graylevel Images

Do the following.

(a) Generate Compound Image

Image **`dipxe_text.tif`** is available as a part of this assignment. Similarly, image **`cameraman.tif`** is available in the book resourse package. Superimpose image **`dipxe_text.tif`** onto the image **`cameraman.tif`** using the imlincomb function with equal 50% contributions. The image should look like the following.

MATLAB script:

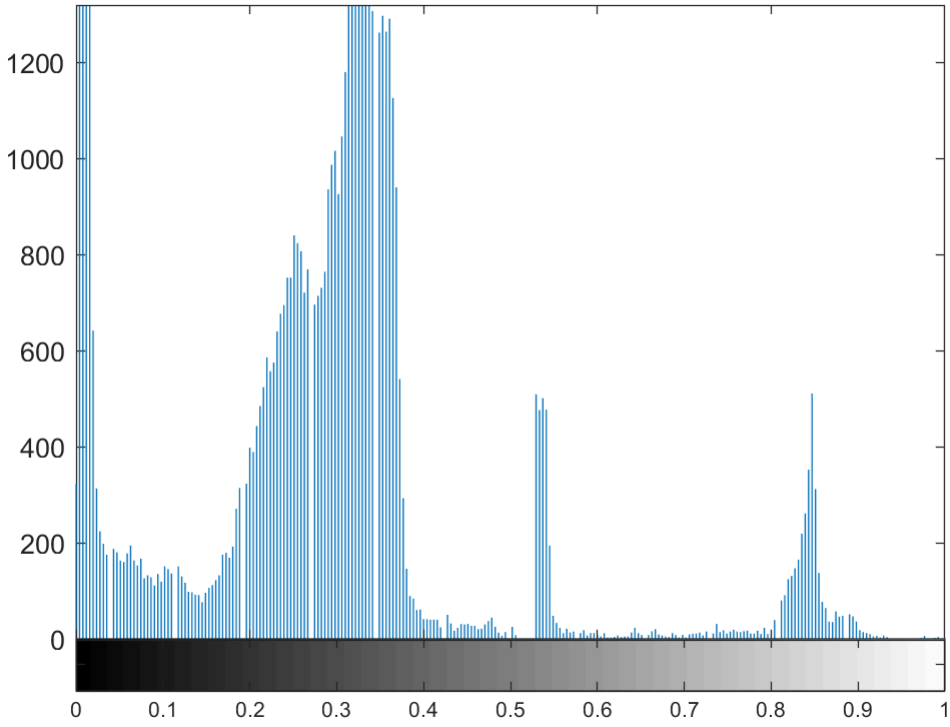
```
clc; close all; clear;
f = im2uint8(imread('dipxe_text.tif'));
g = im2uint8(imread('cameraman.tif'));
h = intensityScaling(imlincomb(.5,f,.5,g));
figure; imshow(h), title('Equally Combined Image')
```

(b) Thresholding

We want to threshold the above compound image to isolate the text '`DIP-XE`'. Can you do it? If yes, then provide the result. If not, explain why not.

MATLAB script:

```
figure, imhist(h)
```




```
compound = (h > .52);
figure, imshow(compound), title('Text Extracted')
```

Looking at the Histogram of the compounded image, we see there is a distinct gap in intensity values around 0.5. Using the value of 0.52 as the threshold for segmentation, we can pull out the original DIP-XE text.

Provide the listing of your functions below.

Function `edgeKernel6H`

```
function w = edgeKernel6H(type,dir)
```

```
% EDGEKERNEL6H spatia1 edge kerne1.
% G = EDGEKERNEL6H(TYPE,DIR) generates a 3-by—3 edge kernel of
% specified TYPE and direction (DIR). These kernels are shown in
% Figs. 10.14 and 10.15 of DIP4E.
%
% Valid TYPE/DIR pairs are:
% TYPE DIR
% 'prewitt' 'v'/'h'
% 'sobe1' 'v'/'h‘
% 'kirsch' 'n'/'nw'/'w'/'sw'/'s'/'se'/'e'/'ne'

% Enter your code below
switch type
    case 'prewitt'
        switch dir
            case 'v'
                w = [-1,0,1;-1,0,1;-1,0,1];
            case 'h'
                w = [-1,-1,-1;0,0,0;1,1,1];
        end
    case 'sobel'
        switch dir
            case 'v'
                w = [-1,0,1;-2,0,2;-1,0,1];
            case 'h'
                w = [-1,-2,-1;0,0,0;1,2,1];
        end
    case 'kirsch'
        switch dir
            case 'n'
                w = [-3,-3,5;-3,0,5;-3,-3,5];
            case 'nw'
                w = [-3,5,5;-3,0,5;-3,-3,-3];
            case 'w'
                w = [5,5,5;-3,0,-3;-3,-3,-3];
            case 'sw'
                w = [5,5,-3;5,0,-3;-3,-3,-3];
            case 's'
                w = [5,-3,-3;5,0,-3;5,-3,-3];
            case 'se'
                w = [-3,-3,-3;5,0,-3;5,5,-3];
            case 'e'
                w = [-3,-3,-3;-3,0,-3;5,5,5];
            case 'ne'
                w = [-3,-3,-3;-3,0,5;-3,5,5];
        end
end
end
```

Function `globalThresh6H`

```
function [g,T,k] = globalThresh6H(f,delT)
% GLOBALTHRESH6H Simple global thresholding.
% [G,T,K] = GLOBALTHRESH6H(F,DELT) iteratively computes a global
```

```
% threshold, T, and uses it to segment image F into two regions
% with values 0 for pixels <= T and 1 otherwise. The segmented
% image is output as G. The algorithm iterates until the difference
% between successive values of T are <= DELT. The final number of
% such iterations is output as K. The program uses the average
% intensity of F as the initial threshold. The value of DELT must
% be a nonnegative integer; it defaults to DELT = 0.01. The
% program normalizes the intensity values of F to the range [0,1].

% Enter your code below
if nargin < 2
    delT = 0.01;
end
if delT < 0
    error('delT must be a nonnegative integer!');
end
k = 0;
f = intensityScaling(f); % scale intensity values of f to range [0,1]
T = mean2(f);
k = 0;
done = false;
while ~done
    k = k + 1;
    g = f > T;
    newT = 0.5*(mean(f(g)) + mean(f(~g)));
    done = abs(T - newT) < delT;
    T = newT;
end
end
```

Function `movingaveragethresh`

```
function g = movingaveragethresh(f,n,K)
% MOVINGAVERAGETHRESH Image segmentation using a moving average threshold.
% G = MOVINGAVERAGETHRESH(F,n,K) segments image F by thresholding its
% intensities based on the moving average of the intensities along
% individua1 rows of the image. The average at pixel k is formed by
% averaging the intensities of that pixel and its n – 1 preceding
% neighbors using Eq. (11-23). To reduce shading bias, the scanning is
% done in a zig–zag manner, treating the pixe1s as if they were a 1-D,
% continuous stream. If the value of the image at a point exceeds K
% percent of the value of the running average at that point, a 1 is
% output in that location in G. otherwise a 0 is output. At the end of
% the procedure, G is thus the thresho1ded (segmented) image. K is a
% sca1ar in the range [0,1].

% Enter your code below
if (K > 1 || K < 0)
    error('K must be a scalar within range [0,1]')
end
[M,N] = size(f);
m = zeros(1,N);
g = false(M,N);
window = n;
```

```
c = (1/window)*ones(1,window);
d = 1;
for k = 1:M
    m(k,:) = filter(c,d,f(k,:));
end
for i = 1:M
    for j = 1:N
        if f(i,j) > K*m(i,j)
            g(i,j) = true;
        else
            g(i,j) = false;
        end
    end
end
end
```