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

Table of Contents

Default Plot Parameters:

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

Problem-6.1: DIP4E Problem 10.14 (Page 868)

Solution: Consider first the 3×3 smoothing kernel mentioned in the problem statement, and a general 3×3 subimage area with intensities a through i, whose center point has value e , as the following figure shows.

1	1	1
1	1	1
1	1	1

Smoothing kernel
(scaled by 9)

a	b	c
d	e	f
g	h	i

Subimage under the
kernel at any one time

Recall that value e is replaced by the response of the 3×3 kernel when its center is at that location. Scaling by 9 so that we can ignore the $1/9$ multiplier in the smoothing kernel, the response of the kernel when centered at that location is $(a+b+c+d+e+f+g+h+i)$.

The idea with the one-dimensional kernel is the same: We replace the value of a pixel by the response of the kernel when it is centered on that pixel. With this in mind, the kernel $[1 \quad 1 \quad 1]$ would yield the following responses when centered at the pixels with values b, e, and h, respectively: $(a+b+c)$, $(d+e+f)$, and $(g+h+i)$. Next, we pass the kernel

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

through these results. When this kernel is centered at the pixel with value e, its response will be

$$\left[(a+b+c)+(d+e+f)+(g+h+i)\right],$$

which is the same as the result produced by the 3×3 smoothing mask.

Returning now to problem at hand, when the g_x Sobel kernel is centered at the pixel with value e, its response is

$$g_x = (g+2h+i) — (a+2b+c).$$

If we pass the one-dimensional differencing kernel

$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

through the image, its response when its center is at the pixels with values d, e, and f, respectively, would be: $(g-a)$, $(h-b)$, and $(i-c)$. Next we apply the smoothing mask $[1 \quad 2 \quad 1]$ to these results. When the kernel is centered at the pixel with value e, its response would be $\left[(g-a)+2(h-b)+(i-c)\right]$, which is $\left[(g+2h+i)-(a+2b+c)\right]$. This is the same as the response of the 3×3 Sobel kernel for g_x. We can use the same basic approach for g_y. However, the directions of the one-dimensional kernel would be reversed in the sense that the differencing kernel would be a column kernel and the smoothing kernel would be a row 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: The first step of the MH algorithm involves filtering of image with an $n \times n$ Gaussian lowpass kernel obtained by sampling

$$G(x, y) = \mathrm{e}^{-(x^2+y^2)/2\sigma^2} = \mathrm{e}^{-x^2/2\sigma^2}\, \mathrm{e}^{-y^2/2\sigma^2} = G(x)G(y)$$

which means that the Gaussian kernel is separable. The second step computes the Laplacian of the image resulting from Step 1 using a 3×3, kernel. These two steps are combined into one equation as

$$g(x, y) = \nabla^2\big[G(x, y) \star f(x, y)\big].$$

which is Eq. (10.30) in DIP4E. Using the definition of the Laplacian operator we can express this equation as

$$\begin{aligned} g(x, y) &= \frac{\partial^2}{\partial x^2}\big[G(x, y) \star f(x, y)\big] + \frac{\partial^2}{\partial y^2}\big[G(x, y) \star f(x, y)\big] \\ &= \frac{\partial^2}{\partial x^2}\big[G(x) \star G(y) \star f(x, y)\big] + \frac{\partial^2}{\partial y^2}\big[G(x) \star G(y) \star f(x, y)\big] \end{aligned}$$

where the second step follows from the separability of 2-D convolution (see Problem 10.22) and $\star$ denotes the 2-D convolution. The terms inside the two brackets are the same, so only two convolutions are required to implement them. From DIP4E Section 10.2, the partials may be written as

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) - 2f(x, y) + f(x-1, y)$$

and

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) - 2f(x, y) + f(x, y-1)$$

The first equation can be implemented via convolution with a 1×3 kernel having coefficients $[1 \quad -2 \quad 1]$ and the second with a 3×1 kernel with the same coefficients. Letting ∇_x^2 and ∇_y^2 represent these two kernels, we have the final result:

$$g(x, y) = \nabla_x^2\big[G(x) \star G(y) \star f(x, y)\big] + \nabla_y^2\big[G(x) \star G(y) \star f(x, y)\big]$$

which requires a total of four 1-D spatial 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: If we use the algorithm as stated in the book, convolving an $M \times N$ image with an $n \times n$ kernel will require $n^2 \times M \times N$ multiplications (see the solution to Problem 10.22). Then convolution with a 3×3 Laplacian kernel will add another $9 \times M \times N$ multiplications for a total of $(n^2+9) \times M \times N$ multiplications. Decomposing a 2-D convolution into 1-D passes requires $2nMN$ multiplications, as indicated in the solution to Problem 10.22. Two more convolutions of the resulting image with the 1×3 and the 3×1 derivative kernels adds $3MN + 3MN = 6MN$ multiplications. The computational advantage is then

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

which is independent of image size. For example, if $n = 25$, $A = 11.32$, so it takes on the order of 11 times more multiplications if direct 2-D convolution were used. Of course, the number of multiplications itself is very much dependent on image size.

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.
%
% Vaiid TYPE/DIR pairs are:
% TYPE DIR
% 'prewitt' 'v'/'h'
% 'sobe1' 'v'/'h'
% 'kirsch' 'n'/'nw'/'w'/'sw'/'s'/'se'/'e'/'ne'

% copyright 2017, R. C. Gonzaiez & R. E. woods.

%—Generate kerne1 based on type/dir:
switch type
    case 'prewitt'
        switch dir
            case 'v'
                w = [-1 O 1;-1 O 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;
wKnw = edgeKernel6H('kirsch','nw')
```

```
wKnw = 3×3
    -3     5     5
    -3     0     5
    -3    -3    -3
```

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

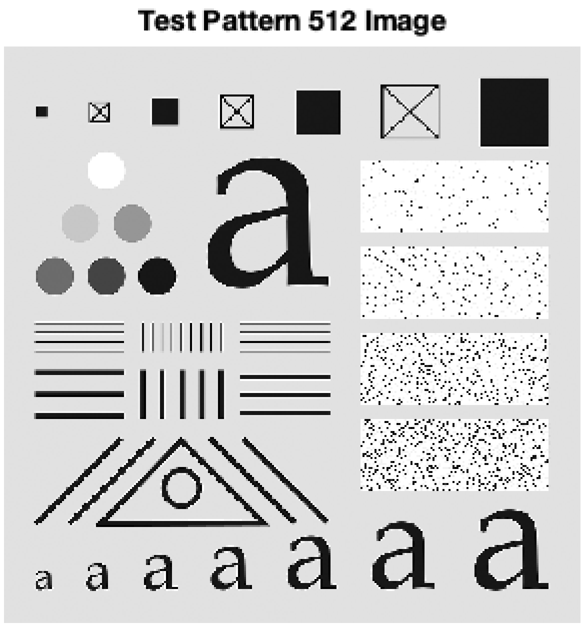
```
wKsw = 3×3
     5    -3    -3
     5     0    -3
     5    -3    -3
```

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

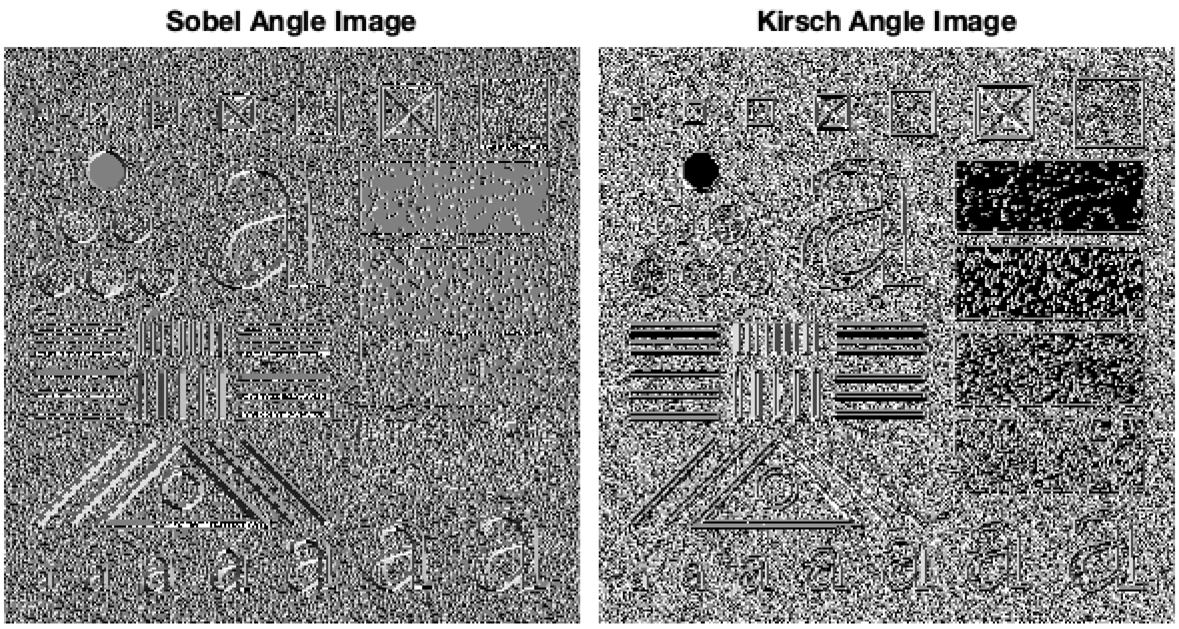
For this part use the **`edgeAngle4e`** function from your DIP4E Student Resources.

MATLAB script:

```
f = imread('../artfiles/testpattern512.tif');
figure('Position',[0,0,4,4.3]*72); subplot('Position',[0,0,1,4/4.3]);
imshow(f); title('Test Pattern 512 Image');
```



```
% Compute and scale the angle images.
angles = edgeAngle4e(f,'sobel'); angles = intScaling4e(angles,'full');
angleK = edgeAngle4e(f,'kirsch'); angleK = intScaling4e(angleK,'full');
figure('Position',[0,0,8.125,4.3]*72); subplot('Position',[0,0,4/8.125,4/4.3]);
imshow(angles); title('Sobel Angle Image');
subplot('Position',[4.125/8.125,0,4/8.125,4/4.3]);
imshow(angleK); title('Kirsch Angle Image');
```



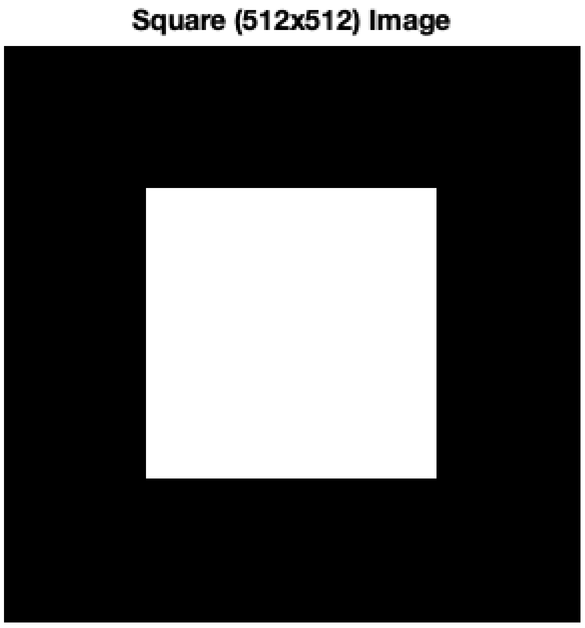
Comment: From the above images, you can see that the edge angles are more consistent using the Kirsch kernels because they have eight possible angle directions, as opposed to four for the Sobel kernels. For a good example of the difference in quality between the two approaches examine the angles along the edge of the large letter 'a'.

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

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

MATLAB script:

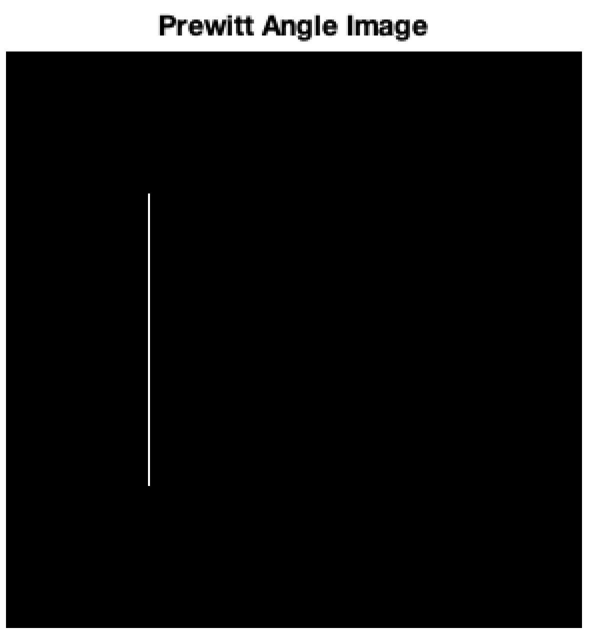
```
% Angles of Square
f = zeros(512); f(128:384,128:384) = 1;
figure('Position',[0,0,4,4.3]*72); subplot('Position',[0,0,1,4/4.3]);
imshow(f); title('Square (512x512) Image');
```

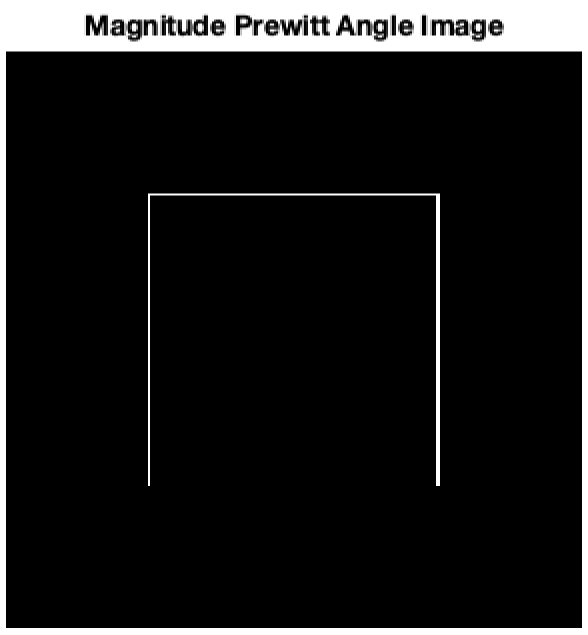


```
af = edgeAngle4e(f,'prewitt');
```

To determine the sign of the angles we will display the filtered image using **`imshow(af)`** which clips negative values, and using **`imshow(abs(af))`** which converts all negative values to positive (which obviously are displayed).

```
figure('Position',[0,0,8.125,4.3]*72);
subplot('Position',[0,0,4/8.125,4/4.3]);
imshow(af); title('Prewitt Angle Image');
subplot('Position',[4.125/8.125,0,4/8.125,4/4.3]);
imshow(abs(af)); title('Magnitude Prewitt Angle Image');
```





Comments: As the above images show, the angle image in fact has negative values. The reason is that, although the object is of constant intensity, its edges are in opposite directions. For example, the vertical edge on the left is black on the left and white on the right, so it is at a positive angle with respect to the way the kernel is defined. The opposite edge is white on the left and black on the right, so the sign of its angle is negative. Note that the bottom edge has zero intensity because it angle has zero degrees.

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].

%—Preliminaries.
%——Scale image to the full [0,1] range.
f = intScaling4e(f);
%——set defaults.
```

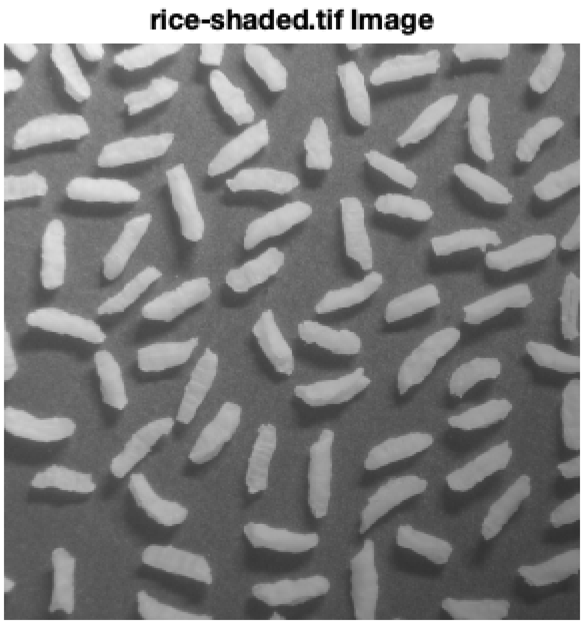
```
if nargin == 1
delT = 0.01;
end

%-start iterating.
k = 0;
%——Initialize.
T = mean(f(:));
Told = inf;
Tnew = T;
%-—Iterate.
while abs(Tnew - Told) > delT
k = k + 1;
G = f > T;
T = 0.5*(mean(f(G)) + mean(f(~G)));
Told = Tnew;
Tnew = T;
end

%—Perform the final segmentation.
g = f > Tnew;

end
```

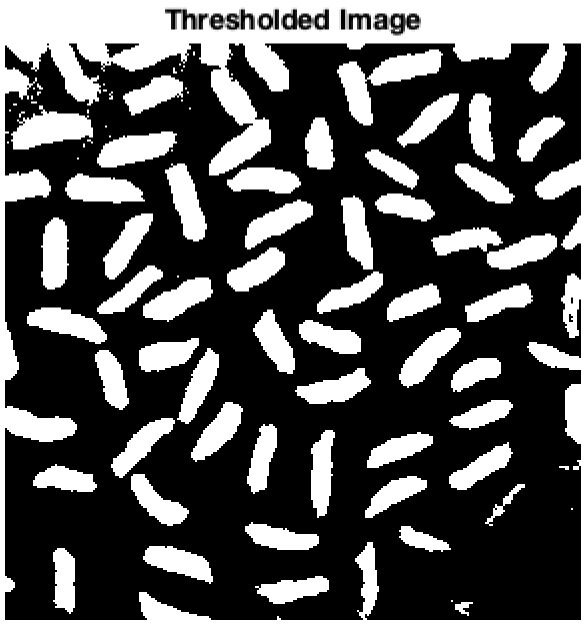
(b) Thresholding of `rice-shaded.tif` Image

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 = imread('../artfiles/rice-shaded.tif');
[g,T,k] = globalThresh6H(f);
figure('Position',[0,0,8.125,4.3]*72);
subplot('Position',[0,0,4/8.125,4/4.3]);
imshow(f); title('rice-shaded.tif Image');
subplot('Position',[4.125/8.125,0,4/8.125,4/4.3]);
imshow(abs(g)); title('Thresholded Image');
```





```
display(k), display(T)
```

```
k = 4
T = 0.5236
```

The algorithm convered in four iterations to a value of T = 0.5236. As the above thresholded shows, global thresholding did a reasonable job of extracting the objects from the background. There are several errors in the bottom right of the image, where object pixels were classified as background. These errors were caused by the fact that there is an illumination gradient that biases the intensity of some of the objects below the average intensity of the image. The opposite situation occurs in the upper left, where some background pixels were classified as object pixels.

(c) Better Value of Parameter delT?

MATLAB script: The parameter **`delT`** determines the difference during iteration between the new and previous estimates of the threshold value. The smallest value that difference can be is 0. The default value for **`delT`** is 0.01. The only other value we need to try is delT = 0.

```
[g0,T0,k0] = globalThresh6H(f,0);
display(k0), display(T0)
```

```
k0 = 6
T0 = 0.5251
```

As we can see, it took two more iterations for convergence, but the thresholds are almost identical, so we could not improve.

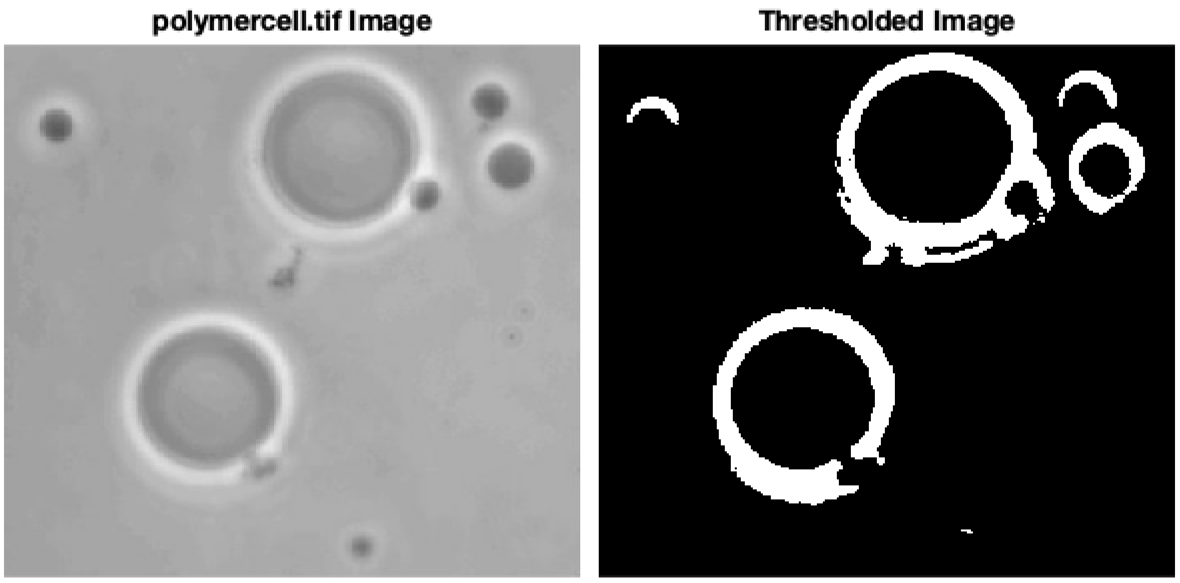
Problem-6.5: Otsu Global Thresholding

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('../artfiles/polymercell.tif');
[g,sep,T] = otsuThresh4e(f);
figure('Position',[0,0,8.125,4.3]*72);
subplot('Position',[0,0,4/8.125,4/4.3]);
imshow(f); title('polymercell.tif Image');
subplot('Position',[4.125/8.125,0,4/8.125,4/4.3]);
imshow(abs(g)); title('Thresholded Image');
```



Confirmation: The segmentation result above is identical to the result in Fig. 10.39 of DIP4E.

```
disp(sep), disp(T)
```

```
    0.5213
   182
```

The separability measure is slightly different than the result in the book, but the thresholds are identical. The difference in the separability measure is due to differences in implementation (the resuit in the book was obtained using function **`grayThresh`** in the Image Processing Toolbox).

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

MATLAB script: We will compare thresholded images using both functions: **`globalThresh6H`** and **`otsuThresh4e`**.

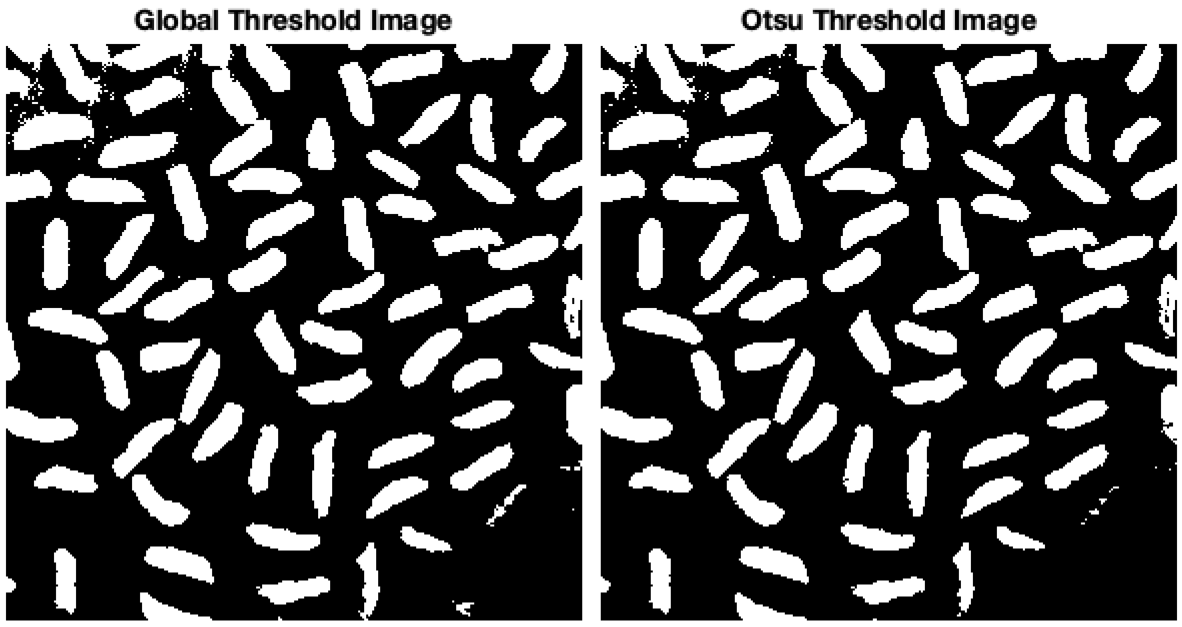
```
f = imread('../artfiles/rice-shaded.tif');
[gG,TG] = globalThresh6H(f); TG = TG*256; display(TG)
```

```
TG = 134.0454
```

```
[gO,sep,TO] = otsuThresh4e(f); display(TO)
```

```
TO = 135
```

```
figure('Position',[0,0,8.125,4.3]*72);
subplot('Position',[0,0,4/8.125,4/4.3]);
imshow(gG); title('Global Threshold Image');
subplot('Position',[4.125/8.125,0,4/8.125,4/4.3]);
imshow(gO); title('Otsu Threshold Image');
```

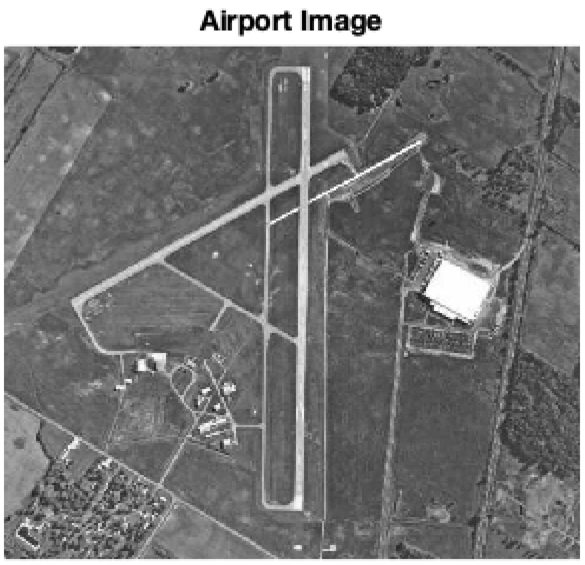


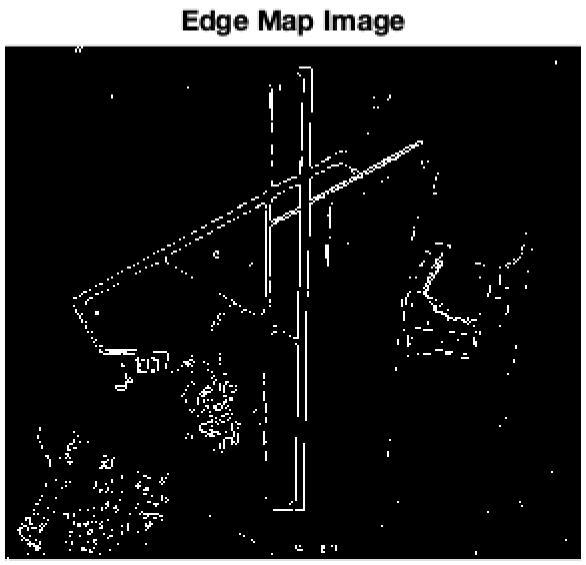
Comments: As the following image and threshold show, the results of the two thresholding approaches are the same. The reason that the Otsu method did not improve the result is that both approaches are global, and the intensity shading of the rice image is not correctable by global thresholding. In other applications the two methods can give quite different results, with the Otsu approach generally being superior.

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.

```
clc; close all; clear;
f = im2double(imread('../artfiles/airport.tif'));
g = edge(f,'sobel',0.15,'nothinning');
figure('Position',[0,0,8.125,4.3]*72);
subplot('Position',[0,0,4/8.125,4/4.3]);
imshow(f); title('Airport Image');
subplot('Position',[4.125/8.125,0,4/8.125,4/4.3]);
imshow(g); title('Edge Map Image');
```

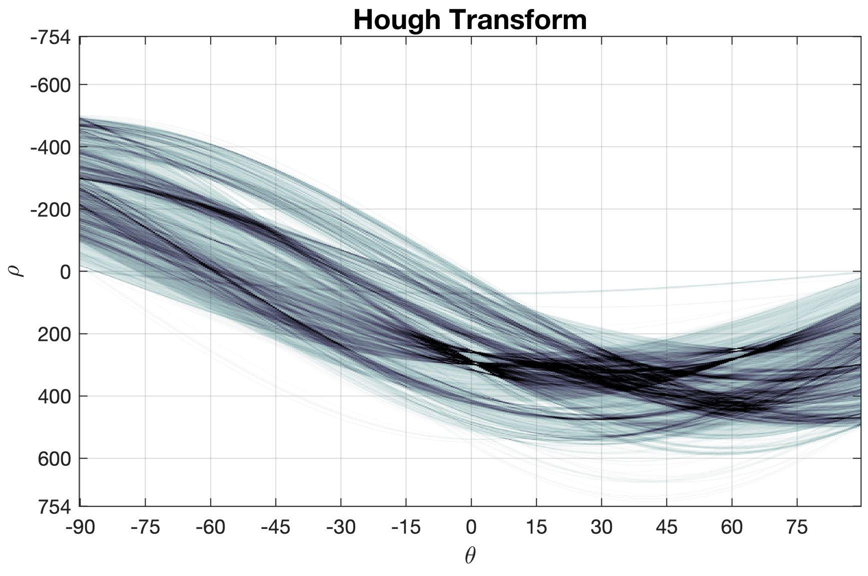




(a) DIPUM3E MATLAB Project 11.4 (b) (Page-718).

MATLAB script:

```
% Compute the Hough transform.
[H,theta,rho] = hough(g,'rhoResolution',2,'thetaResolution',0.5);
% Hough transform disp1ayed as an image:
Him = imadjust(rescale(H)); % Enhance.
[M,N] = size(Him);
% figure("Position",[0,0,(N/M)*4+0.2,4.6]*72,"Color",0.9*[1,1,1]);
% subplot('Position',[0.1/((N/M)*4+0.2),0.2/4.6,(N/M)*4/((N/M)*4+0.2),4/4.6]);
figure("Position",[0,0,7,4]*72);
imagesc(theta,rho,imcomplement(Him)); colormap("bone");
xlabel('\theta'); ylabel('\rho'); title('Hough Transform');
set(gca,'xtick',theta(1):15:theta(end),...
    'ytick',sort([rho(1),rho(end),(-800:200:800)])); grid;
```



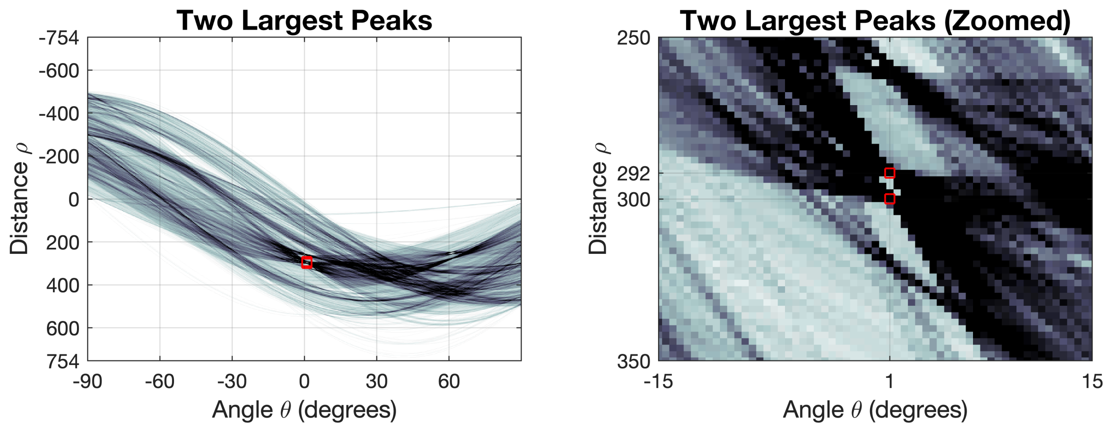
(b) DIPUM3E MATLAB Project 11.4 (c) (Page-718).

MATLAB script:

```
% Extract the 2 most significant peaks using athreshold of
% 0.75*max(H(:)), which is determined experimentally.
peaks = houghpeaks(H,2,'Threshold',0.75*max(H(:)),'NHoodsize',[3,3]);
```

The detected two most significant peaks are superimposed as red squares and are shown below in two subplots. The first subplot shows the superimposed two peaks on the entire Hough transform while the second one shows them in the zoomed plot, since the peaks are very close to each other. Also note that the **`peaks`** variable gives coordinates into the **`thera`** and **`rho`** variables. Study how these variables are used and also how **`imagesc`** display function is used instead of the **`imshow`** function.

```
figure("Position",[0,0,9,3]*72);
subplot(1,2,1); % Peaks on enire transform
imagesc(theta,rho,imcomplement(Him)); colormap("bone");
xlabel('Angle \theta (degrees)'); ylabel('Distance \rho'); axis([-90,90,-754,754]);
set(gca,'xtick',theta(1):30:theta(end),...
    'ytick',sort([rho(1),rho(end),(-800:200:800)])); grid; hold on;
plot(theta(peaks(:,2)),rho(peaks(:,1)),'LineStyle',"none","Marker","square",...
    "LineWidth",2,"Color",'r');title('Two Largest Peaks',"Color",'k');
subplot(1,2,2); % Zoomed display
imagesc(theta,rho,imcomplement(Him)); colormap("bone"); hold on;
xlabel('Angle \theta (degrees)'); ylabel('Distance \rho');
set(gca,'xtick',sort([-15,15,theta(peaks(2,2))]),...
    'ytick',sort([250,350,rho(peaks(1,1)),rho(peaks(2,1))]));
plot(theta(peaks(:,2)),rho(peaks(:,1)),'LineStyle',"none","Marker","square",...
    "LineWidth",2,"Color",'r');axis([-15,15,250,350]);
title('Two Largest Peaks (Zoomed)',"Color",'k'); grid; hold off;
```



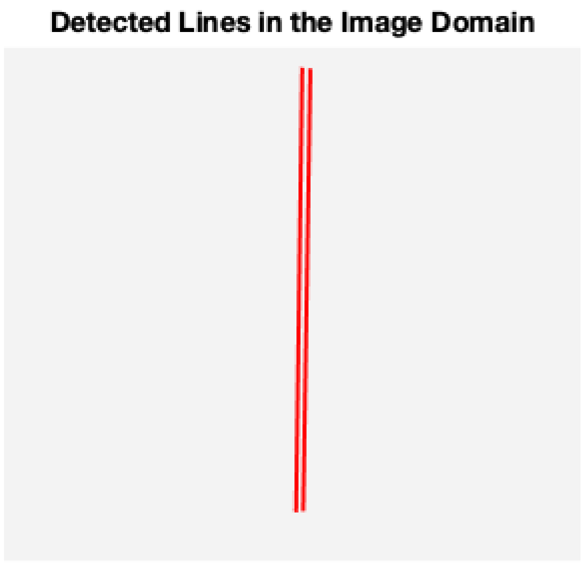
(c) DIPUM3E MATLAB Project 11.4 (d) (Page-718).

MATLAB script:

```
% Obtain the Hough lines. All lines with gaps up to 25
% pixels are joined
lines = houghlines(g,theta,rho,peaks,'fillgap',25);
s = 0.95*ones(size(f));
```

We first display the lines on a blank background that is of the same size as the image. Thus, we inverse transform results from the problem part (b) into image space.

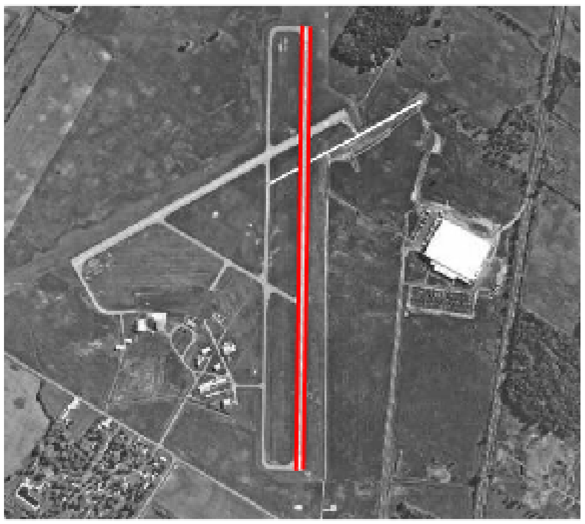
```
figure('Position',[0,0,4,4.3]*72);
subplot('Position',[0,0,1,4/4.3]); imshow(s);hold on;
for k =1:length(lines)
    xy = [lines(k).point1;lines(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,"Color",'r')
end
title('Detected Lines in the Image Domain');
```



Now we superimpose detected lines on original image which clearly identifies the vertical runway.

```
figure('Position',[0,0,4,4.3]*72);
subplot('Position',[0,0,1,4/4.3]); imshow(f);hold on;
for k =1:length(lines)
    xy = [lines(k).point1;lines(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,"Color",'r')
end
title('Detected Lines on the Original Image');
```

Detected Lines on the Original Image



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].

% Preliminaries.
f = tofloat(f);
[M,N] = size(f);
if (n < 1) || (rem(n, 1) ~= 0)
    error('n must be an integer >= 1.')
end
if K < 0 || K > 1
    error('K must be a fraction in the range [0, 1].')
end

% Flip every other row of f to produce the equivalent of a zig—zag
% scanning pattern. Convert image to a vector.
f(2:2:end,:) = fliplr(f(2:2:end,:));
```

```
    f = f'; % still a matrix.
    f = f(:)'; % Convert to row vector for use in MATLAB function filter.

    % compute the moving average.
    maf = ones(1,n)/n; ma = filter(maf,1,f);

    % Perform threshoiding.
    g = f > K * ma;

    % Go back to image format (indexed subscripts).
    g = reshape(g,N,M)';

    % F1ip a1ternate rows back.
    g(2:2:end,:) = fliplr(g(2:2:end,:));

    end
```

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

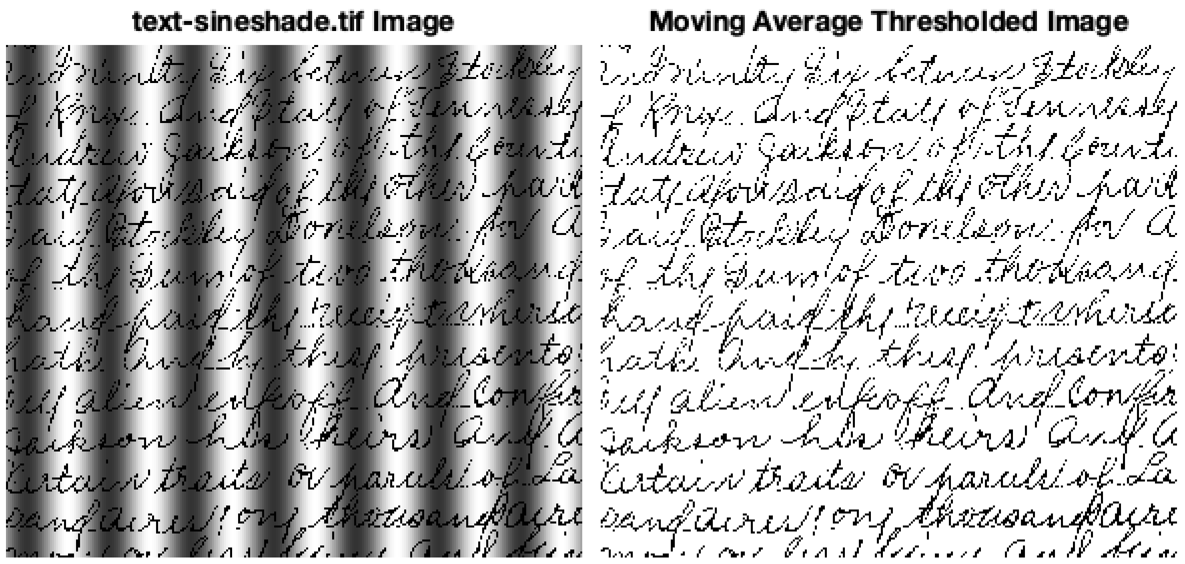
Segment the image **`text-sineshade.tif`**.

MATLAB script: Insert your code below.

```
clc; close all; clear;
f = imread('../artfiles/text-sineshade.tif');
```

Using the reasoning given in Project 11.6(e), that is, that the moving average algorithm is not particularly sensitive to the choice of parameters, we will use those same parameters and check results.

```
[g] = movingaveragethresh(f,20,0.5);
figure('Position',[0,0,8.125,4.3]*72);
subplot('Position',[0,0,4/8.125,4/4.3]);
imshow(f); title('text-sineshade.tif Image');
subplot('Position',[4.125/8.125,0,4/8.125,4/4.3]);
imshow(g); title('Moving Average Thresholded Image');
```



The image on the right above shows that the sine-shaded test was properly segmented.

Problem-6.8 Thresholding of Compound Graylevel Images

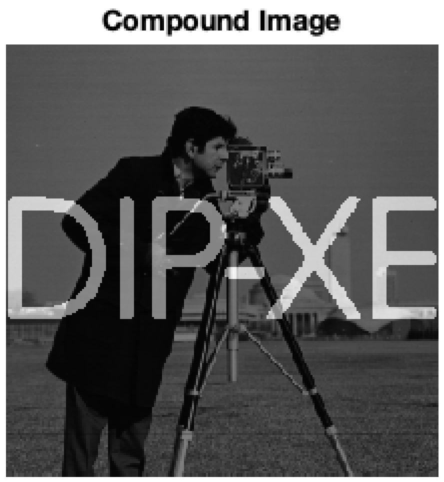
Do the following.

(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.

MATLAB script:

```
clc; close all; clear;
ft = im2uint8(imread('../artfiles/dipxe_text.tif'));
fc = imread('../artfiles/cameraman.png');
f = imlincomb(0.5,ft,0.5,fc);
figure('Position',[0,0,3,3.3]*72); subplot('Position',[0,0,1,3/3.3]);
imshow(f); title('Compound Image');
```

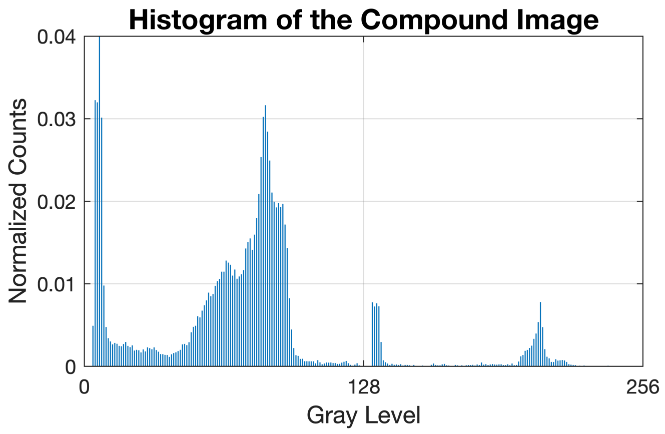


(b) Thresholding

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.

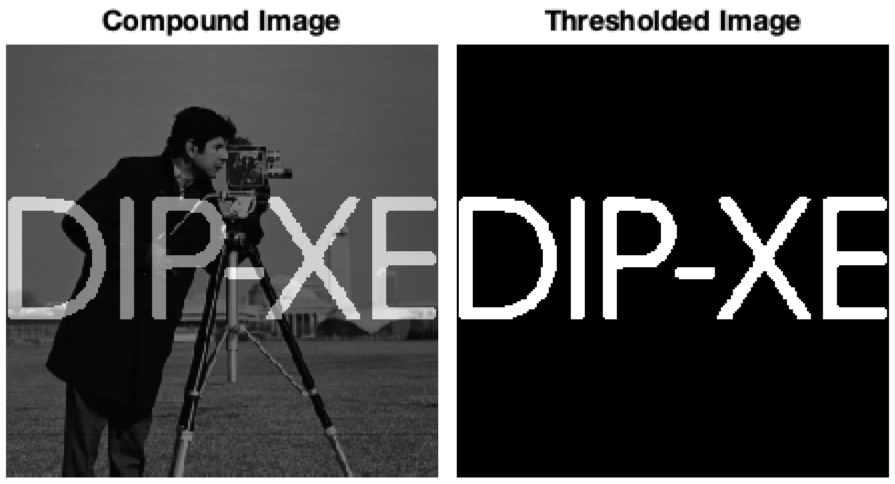
Solution: We will compute and plot histogram of the compound image to determine the value of the appropriate threshold.

```
[counts,bins] = imhist(f,256);
figure('Position',[0,0,5,3]*72);
bar(bins,counts/numel(f),0.5); axis([0,256,0,0.04]);
xlabel('Gray Level'); ylabel('Normalized Counts');
set(gca,'xtick',[0,128,256],'ytick',(0:0.01:0.04));
title('Histogram of the Compound Image'); grid;
```



The compound image **f** and its histogram are shown above. From the histogram we note that the **cameraman** image histogram is below gray level 128 since it is multiplied by 0.5 in the compound image. The **DIPXE** image has only two gray levels; 0 and 255. Adding 50% of it to the **cameraman** image creates an approximate copy of its histogram above level 128. Therefore, $T = 128$ is the proper choice of the needed threshold.

```
T = 128; g = f > T;
figure('Position',[0,0,6.125,3.3]*72);
subplot('Position',[0,0,3/6.125,3/3.3]);
imshow(f); title('Compound Image');
subplot('Position',[3.125/6.125,0,3/6.125,3/3.3]);
imshow(g); title('Thresholded Image');
```



Clearly, the text has been isolated.