EECE-5626 (IP&PR) : Homework-4

Due on November 2, 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: Chapters 4 from DIP4E and DIPUM3E.

Table of Contents

Problem-1: DIP4E Problem 5.11 (Page 442)

In answering the following, refer to the contraharmonic filter in Eq (5-26):

(b) Negative Q

Solution:

Salt noise is reprented by some probability of the pixels of the original image $f(x, y)$ being corrupted and substituted with a bright intensity value of $(L-1)$, based on the bit resolution of the original image. When the values of $Q < 0$, the denominator of the contraharmonic mean filter will reduce to the original value of the pixel intensity of original image $f(x, y)$. This process of reducing the pixel intensity values is effective against the salt noise since it randomly corrupts pixel intensities to be at the upper limit of the intensity value range of the image.

(d) Q is Negative 1.

Solution:

When Q is equal to -1 the formula for the contraharmonic mean filter reduces to

$$\hat{f}(x, y) = \frac{\sum g(r, c)^{Q+1}}{\sum g(r, c)^{Q}} = \frac{\sum g(r, c)^{(-1)+1}}{\sum g(r, c)^{-1}} = \frac{\sum g(r, c)^{0}}{\sum \frac{1}{g(r, c)}} = \frac{mn}{\sum \frac{1}{g(r, c)}}$$

Because the Q term in the numerator equates to zero, every pixel in the image $g(x, y)$ will be to the zeroth power, resulting in a value of 1, then due to the summation, the numerator would just be adding together the number of rows and columns together which equates to the dimension of the image, which is $m * n$. The denominator with a value of -1 for Q will be inverse and result in the summation of $\frac{1}{g(x, y)}$. The end result of $Q = -1$ is the exact formula of the harmonic mean filter equation.

(e) Behavior in Constant Intensity

Solution:

The contraharmonic mean filter cannot eliminate both salt-and-pepper noise simultaneously. It is best suited when an image has been corrupted by intensity value at both the upper limit $(L-1)$ and an intensity value of 0. Thus for an image with constant intensity levels, this filter will have no effect and the pixel values will remain the same after applying the filter.

Problem-2: DIP4E Problem 5.32 (Page 444)

Inverse filtering on additive Gaussian noise.

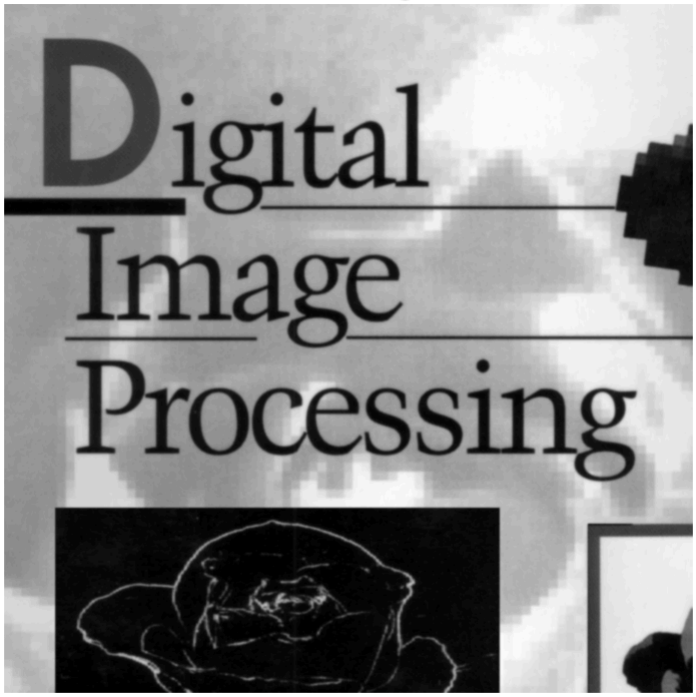
Additional Part: Provide a MATLAB simulation of your explanation to this problem using **`book-cover.tif`** image. Towards this follow the following steps:

1. Read image book-cover.tif and blur it using the parameters of Example 5.8 and the blurring frequency response of Eq. (5.77). Display the blurred image.
2. Inverse filter the blurred image in the frequency domain and display the result. You may need to scale the image for proper display.
3. Generate Gaussian noise with 0 mean and variance equal to 0.0001 and inverse filter it. Display using proper scaling.
4. Add the two images in 2 and 3 and display the result via proper scaling.

Solution:

```
% Part 1 - Blur Book Image by using H(u,v) from Eq(5-77) from DIP4E
f = imread('book-cover.tif'); imshow(f); title('Book Image'); pause(1);
```

Book Image

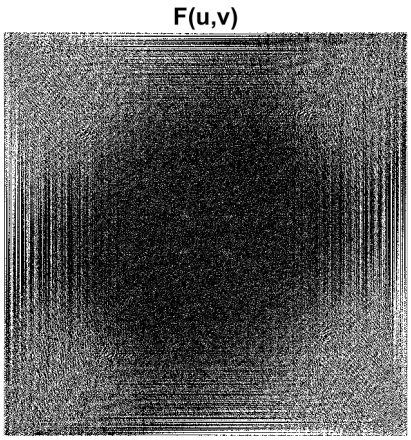


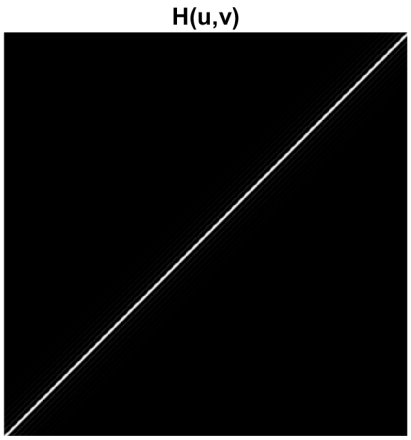

```
% Take FFT of book image and multiply by H
PQ = paddedsize(size(f)); M = PQ(1)/2; N = PQ(2)/2;
[f, revertClass] = tofloat(f);
F = fft2(f);
% Use parameters from Example 5.8 in DIP4E
a = 0.1; b = 0.1; T = 1;
H = motionBlurTF4e(M,N,a,b,T);
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(F); title('F(u,v)','fontsize',14); pause(1);
```

```
Warning: Displaying real part of complex input.
```

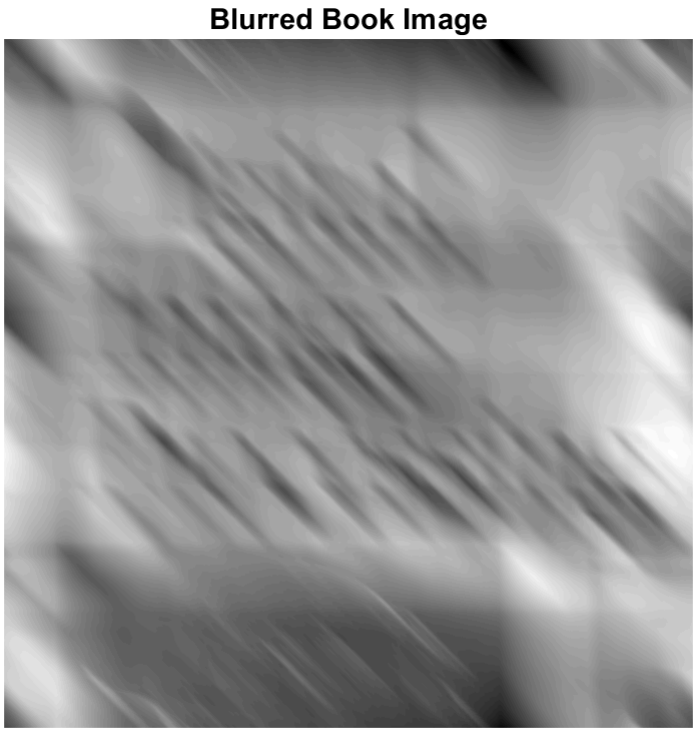
```
subplot(1,2,2); imshow(H); title('H(u,v)','fontsize',14); pause(1);
```

```
Warning: Displaying real part of complex input.
```



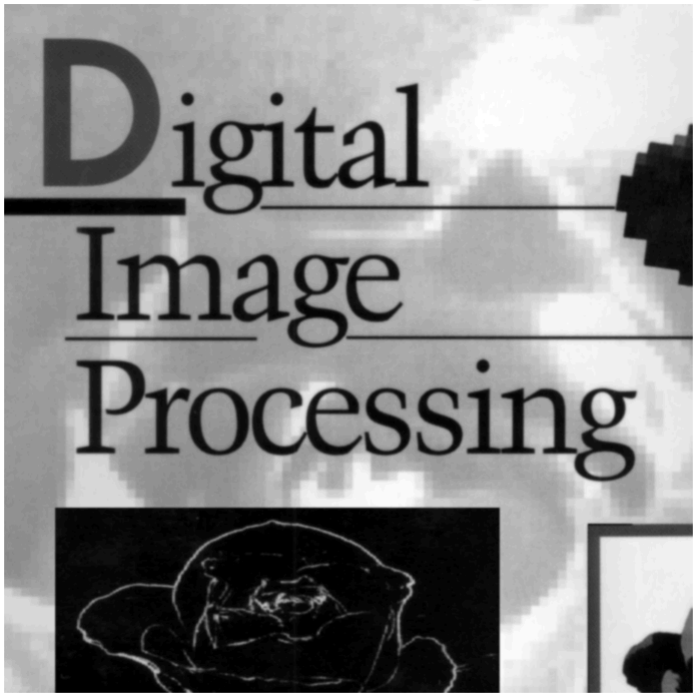



```
G = F.*H;
g = intensityScaling(real(ifft2(G)));
figure; imshow(g); title('Blurred Book Image'); pause(1)
```

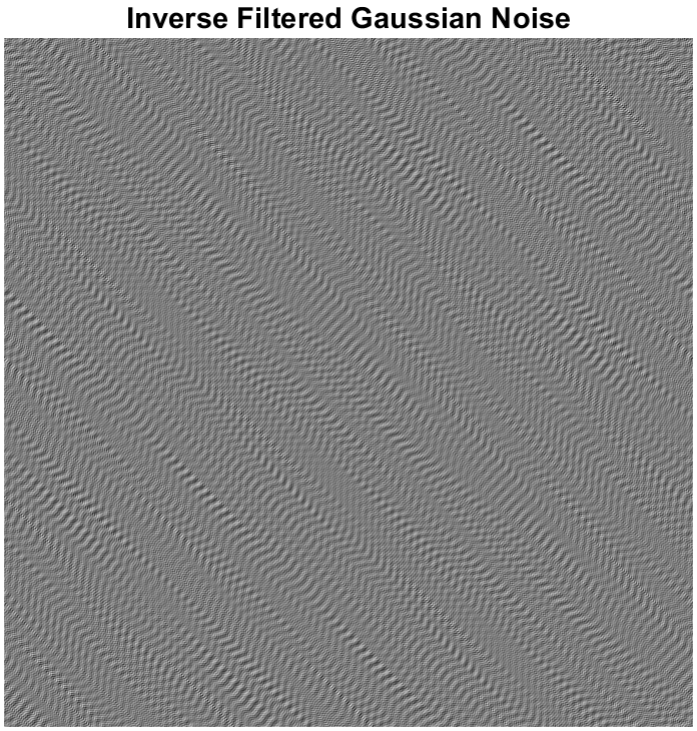



```
% Part 2 - Inverse Filter Blurred Image in FD
HI = 1./H;
f_restored = G.*HI;
f_inv = intensityScaling(real(ifft2(f_restored)));
figure; imshow(f_inv); title("Inverse Filtered Image")
```

Inverse Filtered Image

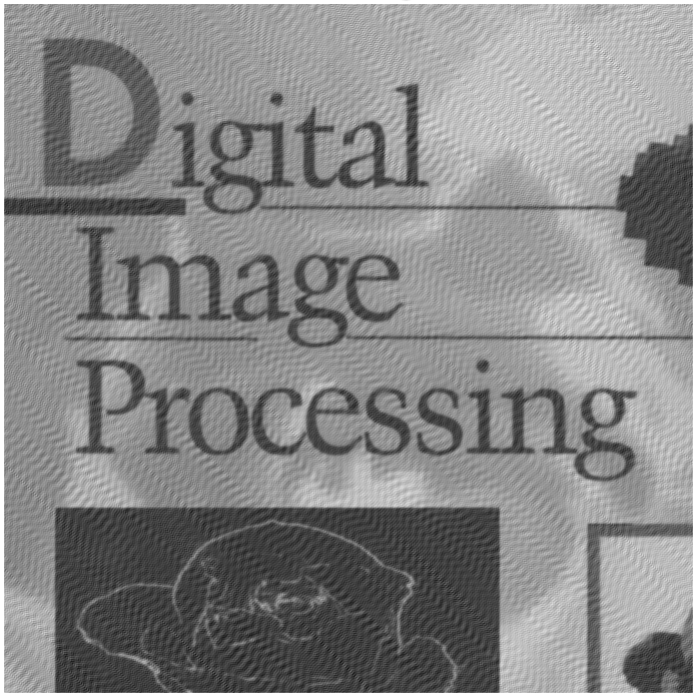



```
% Part 3 - Generate Gaussian noise with mean 0 and variance equal to 0.0001 and inverse filter
GausNoise = 0 + randn([M N])*chol(.0001);
GN = fft2(GausNoise);
f_noise = intensityScaling(real(ifft2(GN.*HI)));
figure; imshow(f_noise); title('Inverse Filtered Gaussian Noise')
```




```
% Part 4 - Add the two images in 2 and 3 and display the result via proper scaling.
f_combined = intensityScaling(double(f_inv) + f_noise);
figure; imshow(f_combined); title("Added Images")
```

Added Images



Problem-3: DIP4E MATLAB Project 5.3 (b) and (e) (Page 446)

Working with salt-and-pepper noise.

(b) MATLAB function `saltPepper4e`

Solution: Insert the code of your function below after the comments.

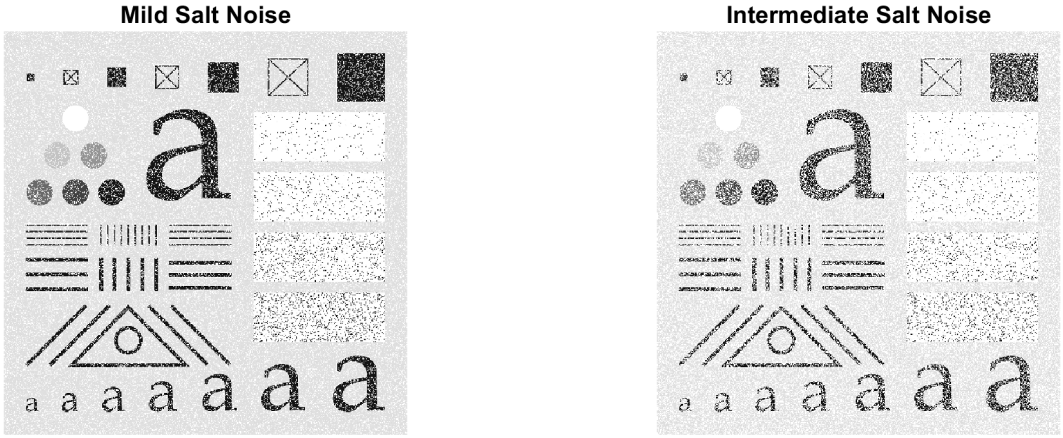
```
function g = saltPepper4e(f,ps,pp)
%SALTPEPPER4E corrupts an image with salt–and-pepper noise.
% g = SALTPEPPER4E(F,PP,Ps) corrupts image F with salt–and–pepper
% noise. Pepper values are 0 with probability PP and salt va1ues
% are 1 with probability PS. Image F can be floating point or can
% have integers values. The output, corrupted image G, is floating
% point with values in the range [0,1]. This function is based on
% implementing Eq. (5-16) in DIP4E.
%

% Insert your code below.
% Convert image to floating-point between range of [0 1]
f = intScaling4e(f);
[M,N] = size(f);
if(ps + pp) > 1
    error('The Sum of PS and PP cannot exceed 1.')
end
X = rand(M,N);
R(1:M,1:N) = 0.5;
```

```
R(X <= pp) = 0;
R(X > pp & X <= ps + pp) = 1;
fn = f;
fn(R==1) = 1; % Salt
fn(R==0) = 0; % Pepper
fn = max(min(fn,1),0);
g = fn;
end
```

(e) Processing of `testpattern512.tif` Image with Salt Noise only

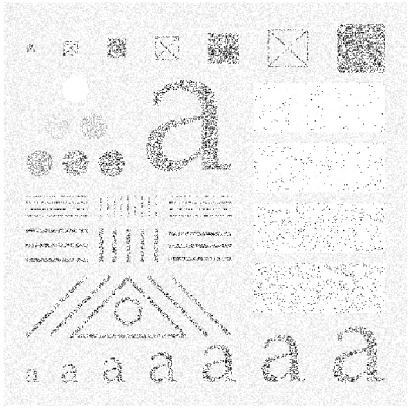
Solution:

```
f = imread('testpattern512.tif');
Nmild = saltPepper4e(f,0.2,0);
Ninter = saltPepper4e(f,0.4,0);
Nheavy = saltPepper4e(f,0.7,0);
Nxtra = saltPepper4e(f,0.9,0);

figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(Nmild); title('Mild Salt Noise','fontsize',14);
subplot(1,2,2); imshow(Ninter); title('Intermediate Salt Noise','fontsize',14);pause(1)
```

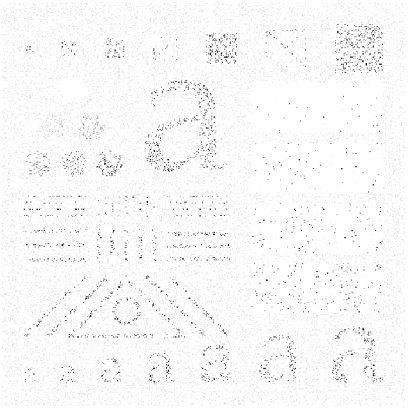



```
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(Nheavy); title('Heavy Salt Noise','fontsize',14);
subplot(1,2,2); imshow(Nxtra); title('Extra Heavy Salt Noise','fontsize',14);pause(1)
```

Heavy Salt Noise



Extra Heavy Salt Noise



Problem-4: DIP4E MATLAB Project 5.9 (Page 449)

Constrained Least Squares (CLS) Filter.

(a) MATLAB Function `constrainedLsTF4e`

Note*:* In the following, parameter **gam** is the same as λ (lambda) that we used in lecture notes.

Solution: Insert the code of your function below after the comments.

```
function L = constrainedLsTF4e(H,gam)
%CONSTRAINEDLSTF4e constrained least squares filter transfer function.
%  L = CONSTRAINEDLSTF4E(H,GAM) implements the constrained least
%  squares filter transfer function defined in Eq. (5-89) of DIP4E.
%  H is the transfer function of the degradation, and GAM is the
%  gamma parameter in Eq. (5-89). This function requires that H be
%  of even dimensions. If H was obtained from a degraded image
%  whose dimensions are not even, delete a row, column, or both, to
%  make the size of the image be even in both directions. Then recompute H.
% Copyright 2017, R. C. Gonzalez & R. E. Woods

% Insert your code below.
[M,N] = size(H);

if(iseven(M) ~= 1)
    H(M,:) = [];
    M = M - 1;
end
if(iseven(N) ~= 1)
    H(:,N) = [];
    N = N - 1;
end
X = zeros(M,N);
p = [0,-1,0;-1,4,-1;0,-1, 0];
X((M/2-1):(M/2+1),(N/2-1):(N/2+1)) = p;
P = abs(fft2(X)).^2;
Habs = abs(H).^2;
Hconj = conj(H);
```

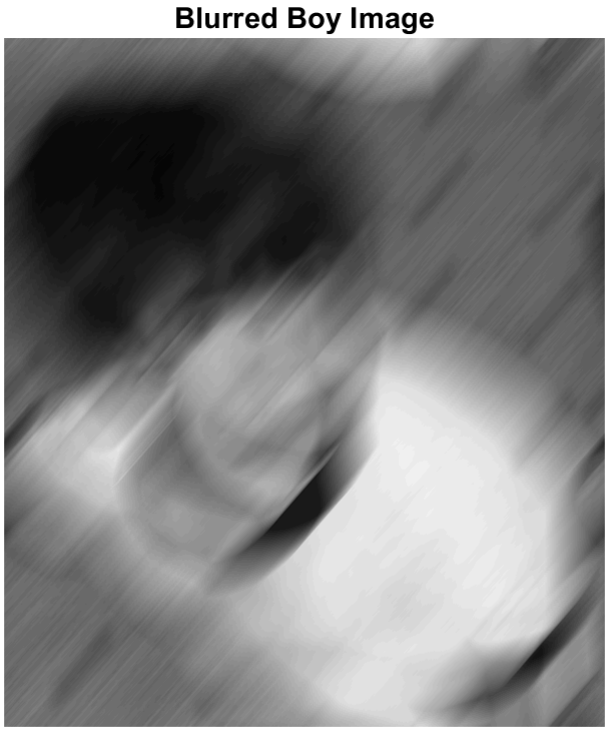
```
    L = (Hconj./(Habs + gam.*P));
    end
```

(b) Restoration of `boy-blurred.tif` Image

Read the image **`boy-blurred.tif`** and restore it using the degradation function from Eq. (5-77).

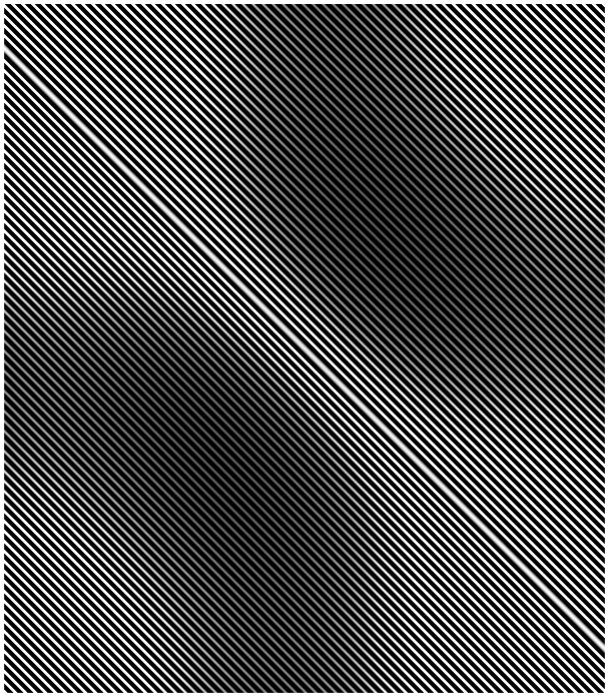
Solution:

```
f = imread('boy-blurred.tif'); figure; imshow(f); title('Blurred Boy Image'); pause(1)
```

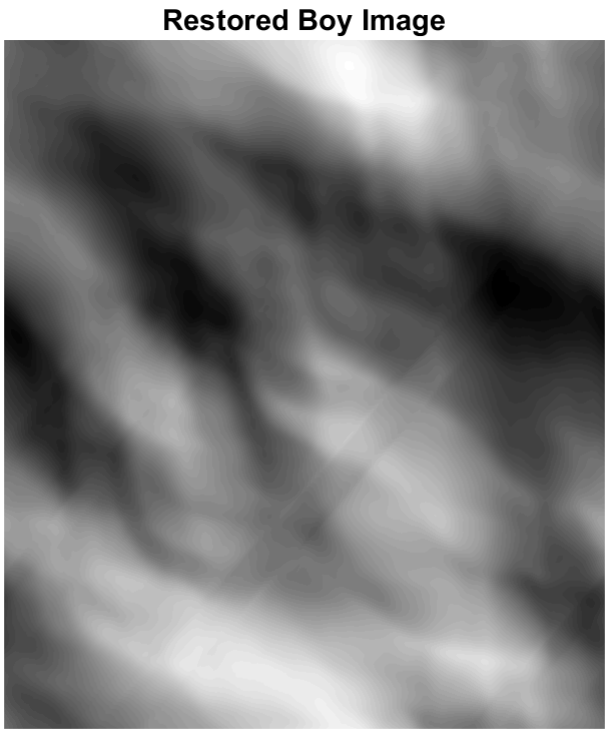



```
% Use parameters from Example 5.8 in DIP4E and Eq 5-77 for transfer
% function H
[M,N] = size(f);
a = 0.1; b = -0.1; T = 1; gamma = 5*10^-4;
H = motionBlurTF4e(M,N,a,b,T);
L = constrainedLsTF4e(H,gamma);
figure; imshow(L);
```

```
Warning: Displaying real part of complex input.
```



```
F = L.*fft2(im2double(f));
f_restored = intensityScaling(real(ifft2(F)));
figure; imshow(f_restored); title('Restored Boy Image')
```



(c) Explanation of results in (b)

Read the image **`boy.tif`** (this was the original before blurring) and explain why your restored image is not quite as good as the original.

Solution:

```
f = imread('boy.tif'); figure; imshow(f); title('Original Boy Image'); pause(1)
```

Original Boy Image

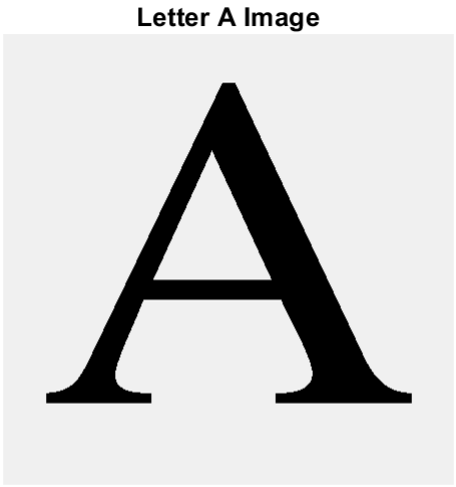
Problem-5: DIPUM3E MATLAB Project 5.7, parts (c) and (d) (Page 318)

Wiener Filtering

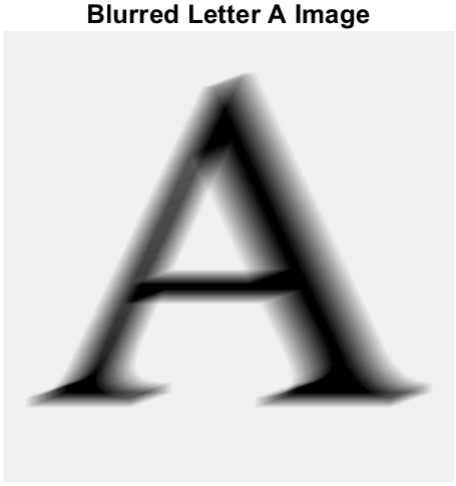
(c) Restoration using Parametric Wiener Filter

Solution:

```
% Read Letter A Image
f = im2double(imread('letterA.tif'));
figure, imshow(f), title('Letter A Image'), pause(1);
```

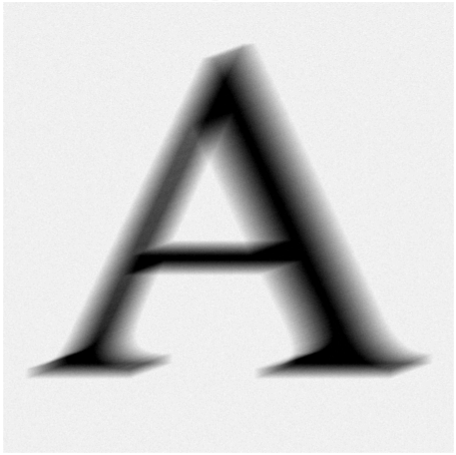



```
% Blur image using motion in the +20 degrees direction for distance of 10%
% of the image width
len = 0.1*size(f,2);
theta = 20;
PSF = fspecial('motion',len,theta);
f_blurred = imfilter(f,PSF,'conv','circular');
figure, imshow(f_blurred), title('Blurred Letter A Image'), pause(1);
```



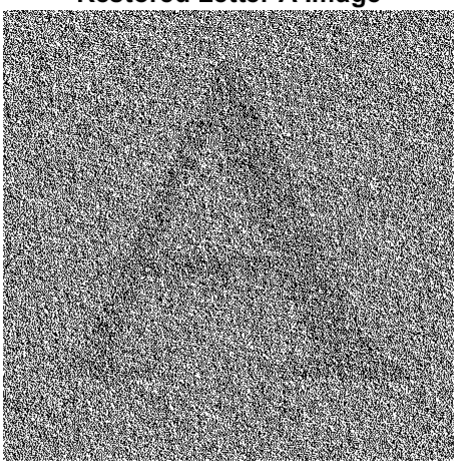

```
% Add Gaussian noise of zero mean and variance of 0.0001
mean = 0;
var = 0.0001;
f_blurred_noise = imnoise(f_blurred,'gaussian',mean,var);
figure, imshow(f_blurred_noise), title('Blurred Letter A Image with Gaussian Noise'), pause(1);
```

Blurred Letter A Image with Gaussian Noise



```
NSPR = 0;
f_blurred_noise = edgetaper(f_blurred_noise,PSF);
frest = deconvwnr(f_blurred_noise,PSF,NSPR);
figure, imshow(frest), title('Restored Letter A Image'), pause(1);
```

Restored Letter A Image



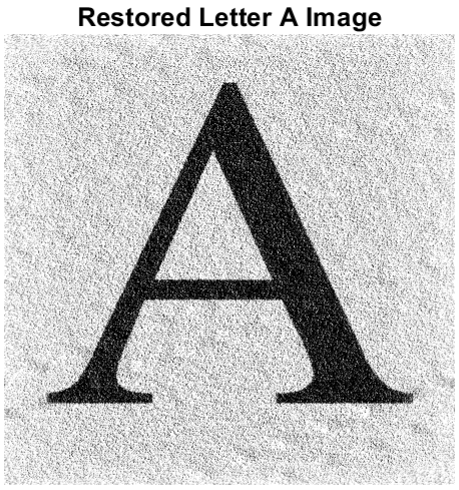
When computing the parametric weiner filter transfer function with the blurred Letter A image, the blur is removed by the image is blanketed by noise since the NSPR was set to zero in the deconvwnr function

(d) Restoration using Noise-to-Signal Power Ratio

Solution:

```
% Find Average Noise Power and Average Image Power to find ratio of NSPR
% then apply Parametric Weiner Filter to degraded image for restoration
noise = f_blurred_noise - f_blurred;
```

```
Sn = abs(fft2(noise)).^2;
nA = sum(Sn(:))/numel(noise);
Sf = abs(fft2(f)).^2;
fA = sum(Sf(:))/numel(f);
R = nA/fA;
f_blurred_noise = edgetaper(f_blurred_noise,PSF);
frest = deconvwnr(f_blurred_noise,PSF,R);
figure, imshow(frest), title('Restored Letter A Image'), pause(1);
```



The image now can be seen clearly as the noise has been minimized after calculating the Average Noise Power and Average Image Power. Using this ratio as a parameter to the input, the resulting image is no longer blurred, but still contains a granular look compared to the original image.

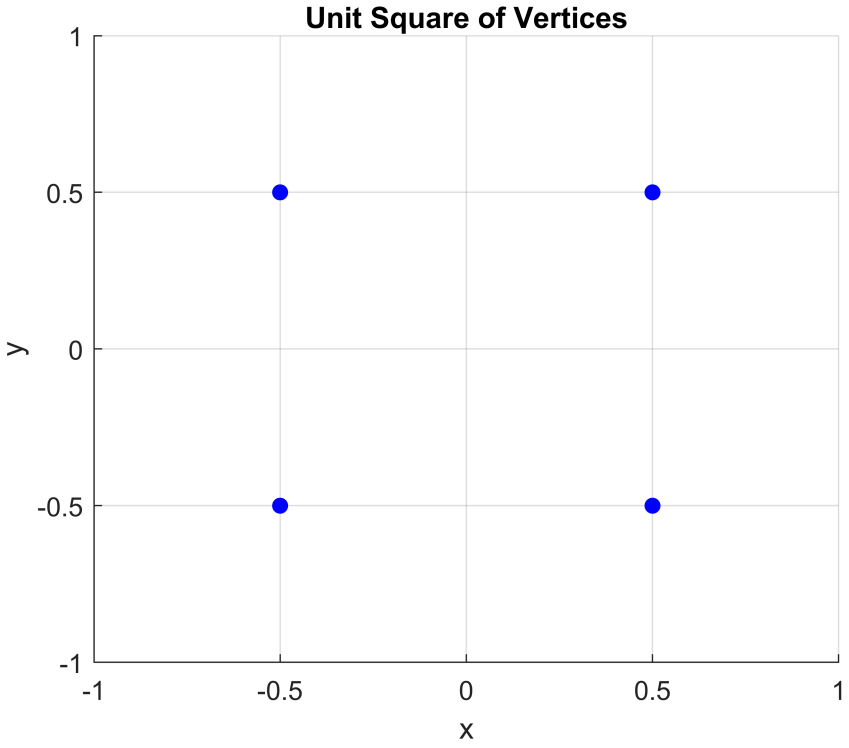
Problem-6: DIPUM3E Project 6.4 (Page 373)

Rotation using Composite Affine Transforms.

(a) Create a Unit Square

Solution:

```
% Make 4 x 2 matrix of unit square centered at the origin. Plot square
% containing vertices
T = [0.5,0.5; -0.5,0.5; -0.5,-0.5; 0.5,-0.5];
x = T(:,1); y = T(:,2);
figure('Units','inches','Position',[0,0,5,4]);
scatter(x,y,'filled','b'); grid ON;
title('Unit Square of Vertices');
xticks(-1:.5:1); yticks(-1:.5:1);
xlabel('x'); ylabel('y');
axis([-1 1 -1 1]); pause(1)
```

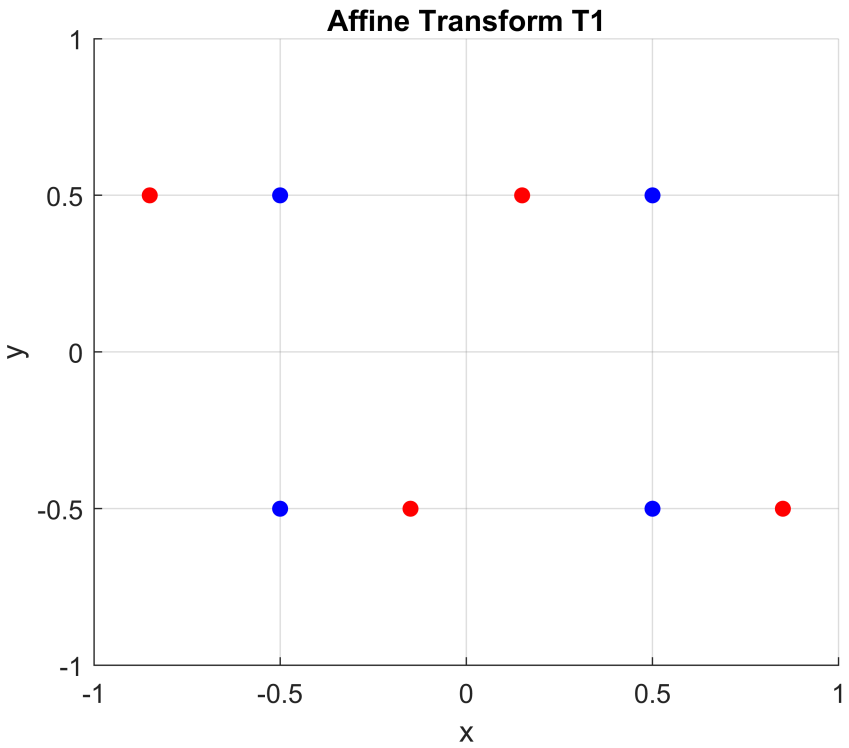


(b) Rotate the Unit Square using Affine Shears

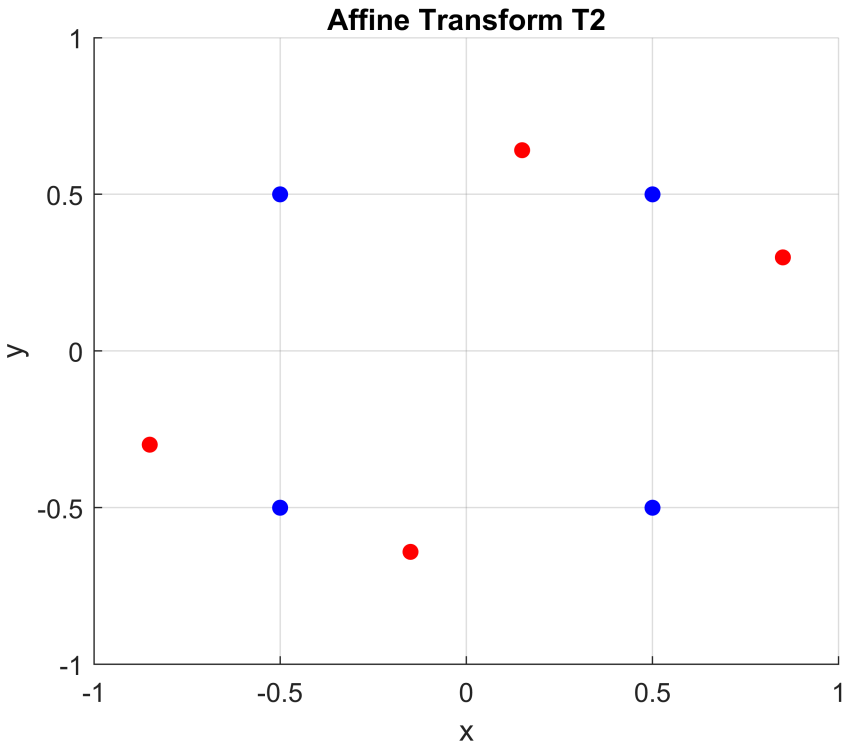
Solution:

```
% Make 3 affine2d objects, tform1, tform2, tform3 with theta = 70 degrees.
% Plot each superimpose of affine2d objects and unit square
theta = 70 * (pi/180);
alpha = -tan(theta/2);
gamma = -tan(theta/2);
beta = sin(theta);

T1 = [1,0,0; alpha,1,0; 0,0,1];
T2 = [1,beta,0; 0,1,0; 0,0,1];
T3 = [1,0,0; gamma,1,0; 0,0,1];

tform1 = affine2d(T1);
tform2 = affine2d(T2);
tform3 = affine2d(T3);

img1 = transformPointsForward(tform1,T);
img1_x = img1(:,1); img1_y = img1(:,2);
figure('Units','inches','Position',[0,0,5,4]);
scatter(img1_x,img1_y,'filled','r');
hold ON;
scatter(x,y,'filled','b'); grid ON;
title('Affine Transform T1');
xticks(-1:.5:1); yticks(-1:.5:1);
xlabel('x'); ylabel('y');
```

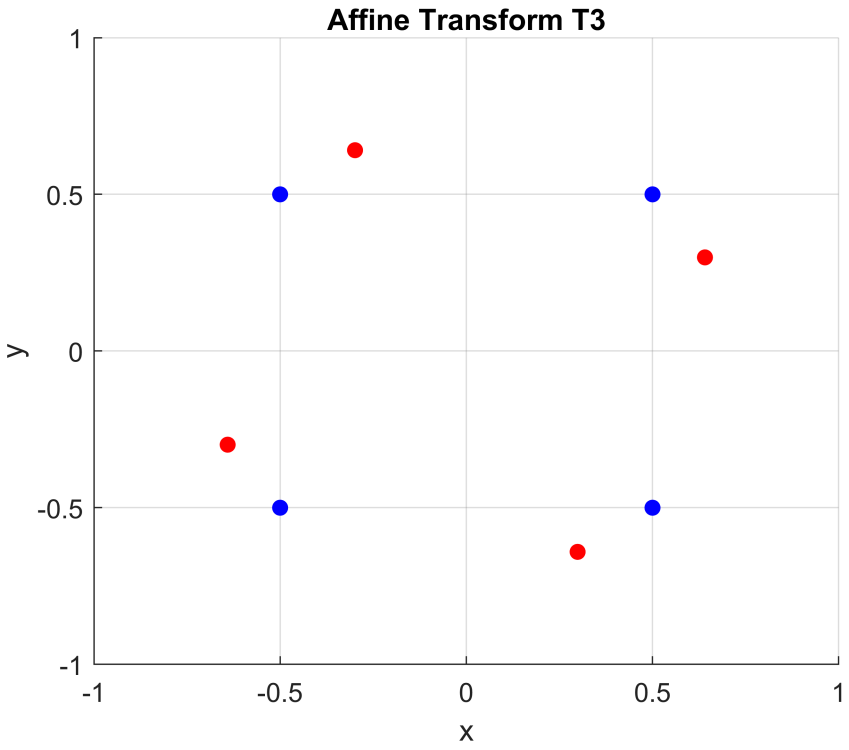
```
axis([-1 1 -1 1]); pause(1)
```




```
img2 = transformPointsForward(tform2,img1);
img2_x = img2(:,1); img2_y = img2(:,2);
figure('Units','inches','Position',[0,0,5,4]);
scatter(img2_x,img2_y,'filled','r');
hold ON;
scatter(x,y,'filled','b'); grid ON;
title('Affine Transform T2');
xticks(-1:.5:1); yticks(-1:.5:1);
xlabel('x'); ylabel('y');
axis([-1 1 -1 1]); pause(1)
```




```
img3 = transformPointsForward(tform3,img2);
img3_x = img3(:,1); img3_y = img3(:,2);
figure('Units','inches','Position',[0,0,5,4]);
scatter(img3_x,img3_y,'filled','r');
hold ON;
scatter(x,y,'filled','b'); grid ON;
title('Affine Transform T3');
xticks(-1:.5:1); yticks(-1:.5:1);
xlabel('x'); ylabel('y');
axis([-1 1 -1 1]); pause(1)
```

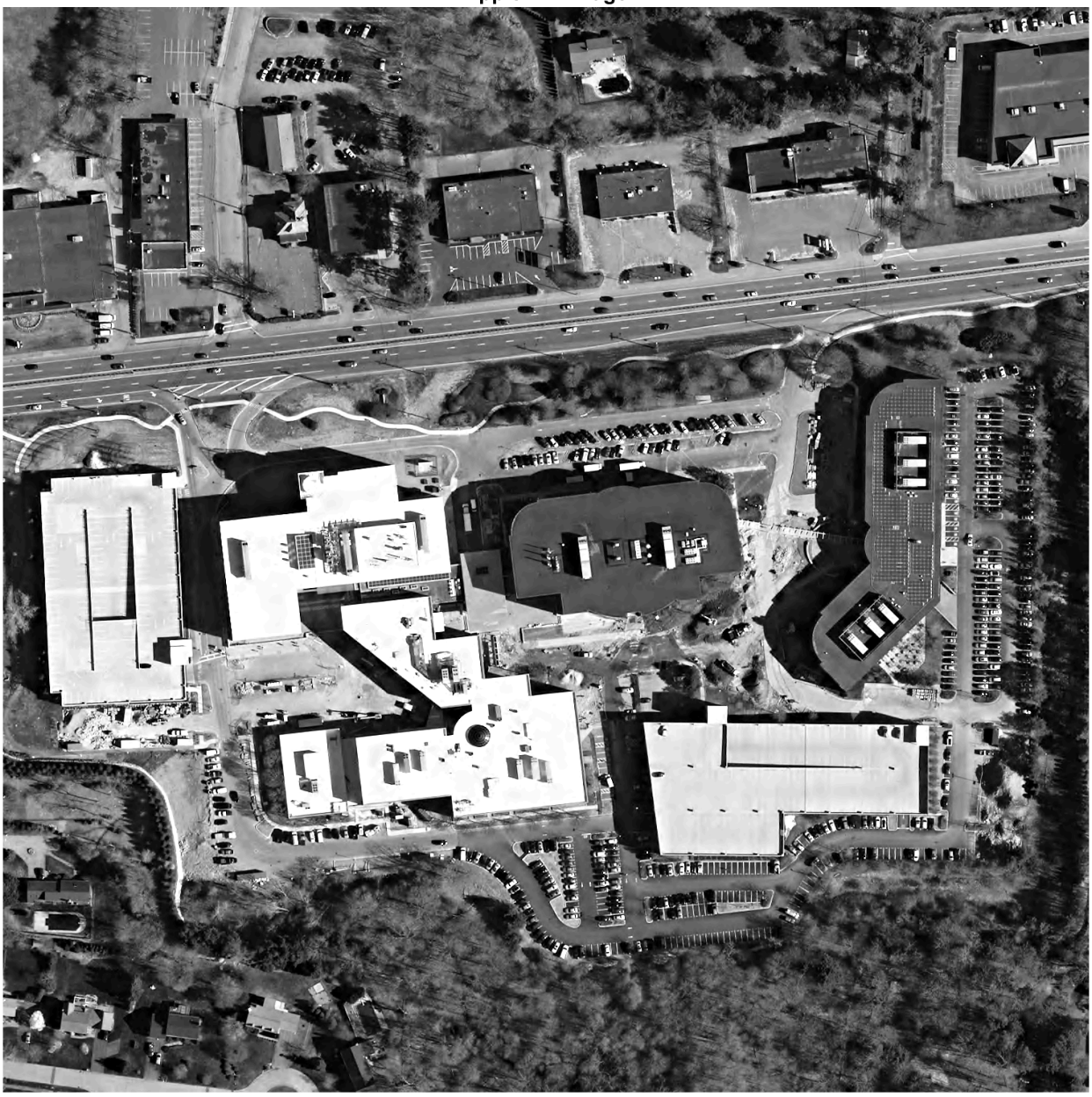


(c) Rotate `apple-hill-2013.png` using Affine Shears

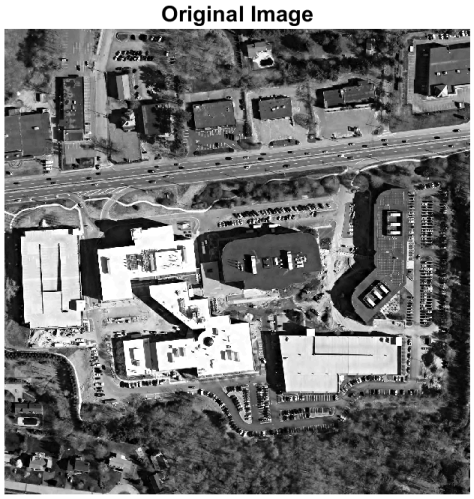
Solution:

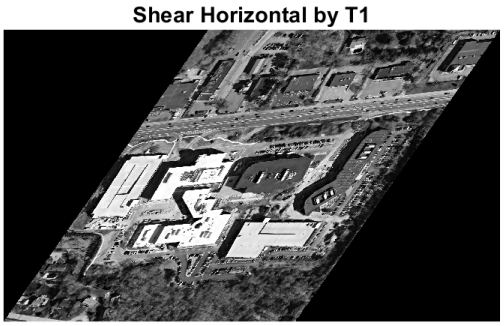
```
% Apply 3 affine2d objects to image 'apple-hill-2013.png and plot results
f = imread('apple-hill-2013.png');
figure('Units','inches','Position',[0,0,5,4]); imshow(f); title('Apple Hill Image');
```

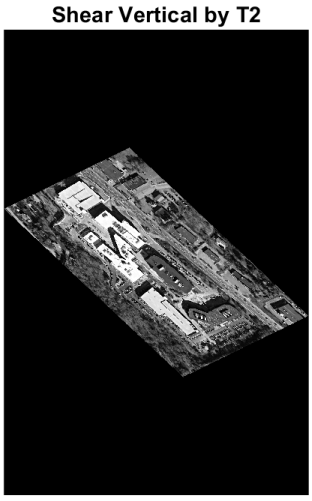
Apple Hill Image

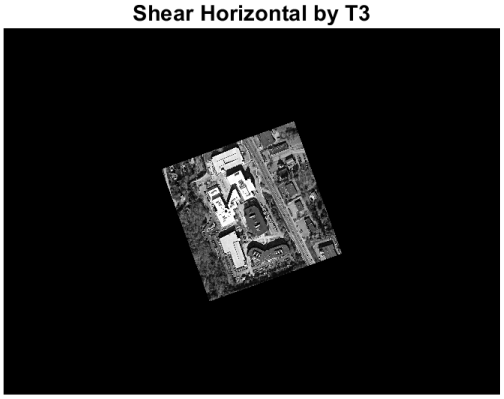


```
g1 = imwarp(f, tform1);
g2 = imwarp(g1, tform2);
g3 = imwarp(g2, tform3);
figure('Units','inches','Position',[0,0,12,28]);
subplot(2,2,1), imshow(f), title('Original Image');
subplot(2,2,2), imshow(g1), title('Shear Horizontal by T1');
subplot(2,2,3), imshow(g2), title('Shear Vertical by T2');
subplot(2,2,4), imshow(g3), title('Shear Horizontal by T3'); pause(1)
```









Problem-7: DIP4E 7.3 (Page 588) and DIP4E 7.25 (a) & (c) (Page 591)

The following two parts, (a) and (b), are not related.

(a) DIP4E 7.3

Study Problem 7.2 to solve this problem.

Solution:

The general expression for the distances between the three valid colors c_1, c_2 and c_3 with given coordinates $(x_1, y_1), (x_2, y_2)$, and (x_3, y_3) and a color c with coordinates (x_0, y_0) that lies within the triangle formed by the three valid colors would be

$d(c, c_1) = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$, $d(c, c_2) = \sqrt{(x_0 - x_2)^2 + (y_0 - y_2)^2}$, and $d(c, c_3) = \sqrt{(x_0 - x_3)^2 + (y_0 - y_3)^2}$

then the distances between each color would be

$d(c_1, c_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, $d(c_1, c_3) = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}$, and $d(c_2, c_3) = \sqrt{(x_2 - x_3)^2 + (y_02 - y_3)^2}$

then the percentages$p$ that each of the colors make up to form the new color c would be

$$p_1 = \frac{d(c_1, c_2) - d(c_1, c_3) - d(c, c_1)}{d(c_1, c_2)} x\ 100$$

$$p_2 = \frac{d(c_1, c_2) - d(c_2, c_3) - d(c, c_2)}{d(c_1, c_2)} x\ 100$$

Percentage p_3 could be derived from $p_3 = 1 - p_1 - p_2$ since the other percentages would be known, or any other combinations where at least two of the percentages are known.

(b) DIP4E 7.25

(a) HSI Components

Solution:

Since the square contains only the colors of Red, Green, and Blue, and these colors contain max saturation and intensity, then the conversion from RGB to HSI values would be

Red = [0, 1, 1]

Green = [1/3, 1, 1]

Blue = [2/3, 1, 1]

When examining the HSI color models, the starting angle for the Hue value starts a 0 degrees which is the Hue value for Red. All of the primary colors are exactly 120 degrees away from each other and since the values are normalized between the range [0 1] the Hue value for Green would be 120 degrees or 1/3 and the Hue value for Blue would be 240 degrees or 2/3. The Saturation and Intensity values are at a max value of 1 for all three HSI components.

(c) Smoothing of Hue component

When applying the 125 x 125 averaging filter kernel to the Hue component of the image, the borders of the colors would start to blur together into colors that lie in between each bordered colors. The borders of Red and Green would start to see a Yellow hue appear while the borders of Blue and Green would start to see a Cyan hue component appear. The center of the image would have a color that is a mixtured of all three colors so the center would start to become a White hue.

Problem-8 DIP4E MATLAB Project 7.2 (Page 592)

Conversion of RGB to Gray Image

(a) MATLAB Function `rgb2gray4e`

Solution: Insert the code of your function below after the comments.

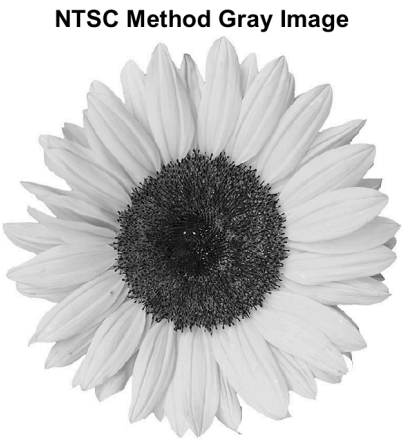
```
function g = rgb2gray4e(f,method)
%RGB2GRAY4E Converts RGB image to grayscale.
% G = RGB2GRAY4E(F,METHOD) converts 24-bit RGB color image,
% F, to an 8-bit grayscale image, G. If METHOD = 'average',
% the average method is used. If METHOD = 'ntsc', the NTSC method
% is used. This is the defauit.

% Insert your code below
if nargin < 2
   method = 'ntsc';
end
f = im2double(f);
R = f(:,:,1);
G = f(:,:,2);
B = f(:,:,3);

switch method
    case 'average'
        g = (R + G + B)/3;
    case 'ntsc'
        g = 0.299*R + 0.587*G + 0.114*B;
end
end
```

(c) Processing of Image `sunflower.tif`

Solution: MATLAB script.

```
f = imread('sunflower.tif');
figure, imshow(f), title('Sunflower Image');
```


```
f_gray_avg = rgb2gray4e(f,'average');
f_gray_ntsc = rgb2gray4e(f,'ntsc');
figure('Units','inches','Position',[0,0,12,4]);
subplot(1,2,1); imshow(f_gray_avg); title('Average Method Gray Image','fontsize',14);
subplot(1,2,2); imshow(f_gray_ntsc); title('NTSC Method Gray Image','fontsize',14);pause(1)
```



The result of using the rgb2gray4e() function shows two differently toned gray images as a result. The averaging method weighed each of the RGB component equally and resulted in a darker grey image. Using the ntsc method resulted in a lighter grayscale image because each of the RGB components were weighted

with specific float values, rather than averaging the values of all three components. This method was used to account for the human color perception, which is more sensitive to green.

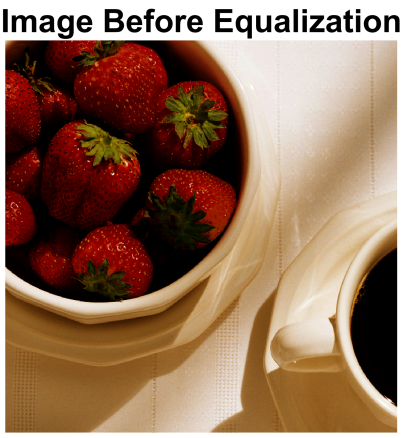
Problem-9 DIP4E MATLAB Project 7.6 (Page 592)

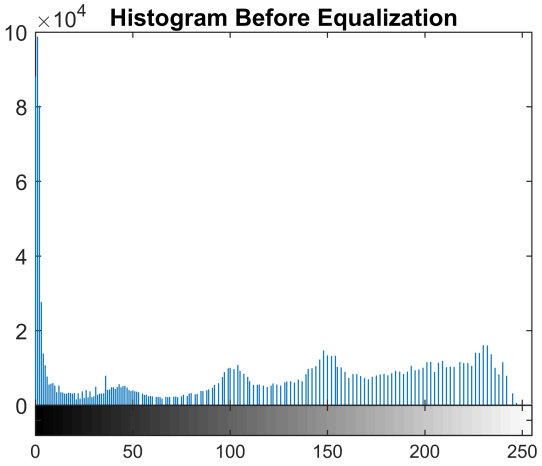
Histogram Equalization

(a) Equalization of strawberries-RGB-dark.tif Image

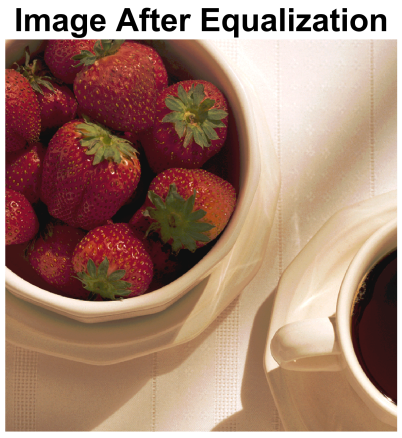
Solution:

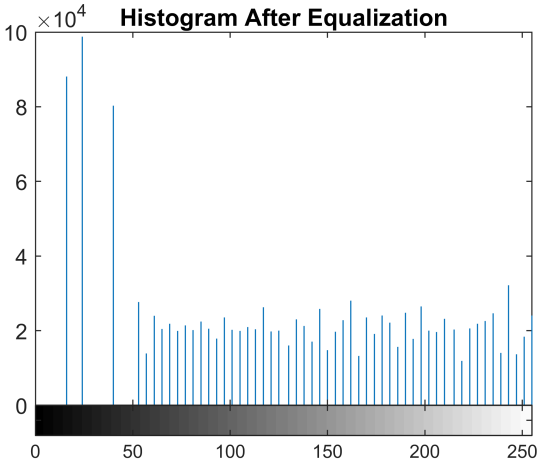
```
f = imread('strawberries-RGB-dark.tif'); imghist = imhist(f); imghistmax = ceil(max(imghist(:))
feq = histeq(f); imghisteq = imhist(feq); imghisteqmax = ceil(max(imghisteq(:))/10000)*20000;
figure('Units','inches','Position',[0,0,9,3]);
subplot(1,2,1); imshow(f); title('Image Before Equalization','fontsize',14);
subplot(1,2,2); imhist(f); set(gca,'ylim',[0,imghistmax]); title('Histogram Before Equalizatio
```






```
figure('Units','inches','Position',[0,0,9,3]);
subplot(1,2,1); imshow(feq); title('Image After Equalization','fontsize',14);
subplot(1,2,2); imhist(feq); set(gca,'ylim',[0,100000]); title('Histogram After Equalization','
```

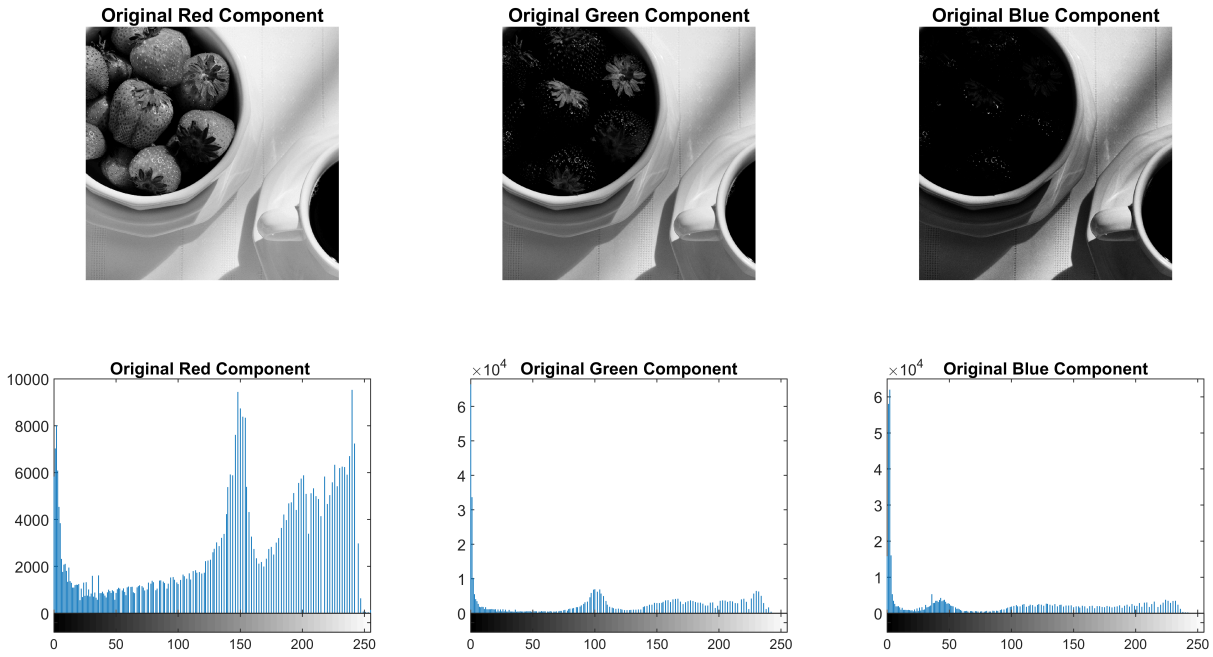




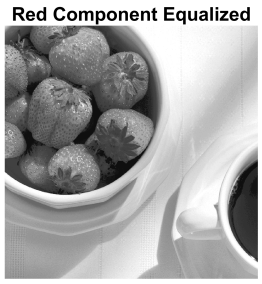
(b) Explanation of Results in (a)

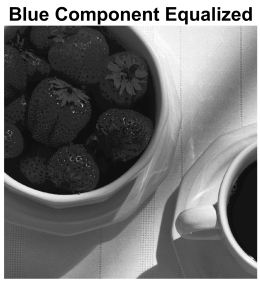
Solution:

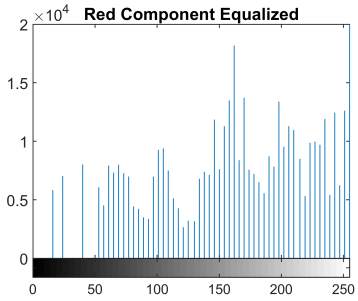
```
% Extract
R = f(:,:,1); G = f(:,:,2); B = f(:,:,3);
figure('Units','inches','Position',[0,0,12,6]);
subplot(2,3,1); imshow(R); title('Original Red Component','fontsize',14);
subplot(2,3,2); imshow(G); title('Original Green Component','fontsize',14);
subplot(2,3,3); imshow(B); title('Original Blue Component','fontsize',14);
subplot(2,3,4); imhist(R); title('Original Red Component ','fontsize',14); ylim([0 10000]);
subplot(2,3,5); imhist(G); title('Original Green Component','fontsize',14); ylim([0 68000]);
subplot(2,3,6); imhist(B); title('Original Blue Component','fontsize',14); ylim([0 65000]);
```

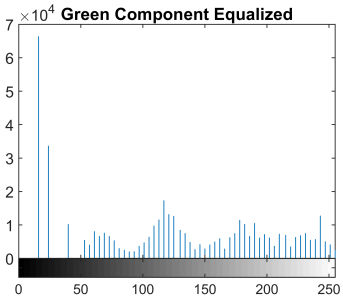


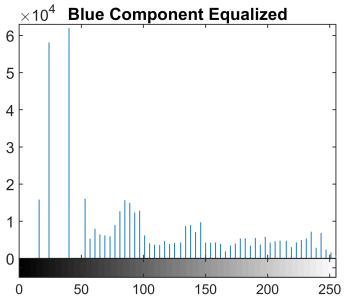

```
Req = feq(:,:,1); Geq = feq(:,:,2); Beq = feq(:,:,3);
figure('Units','inches','Position',[0,0,12,6]);
subplot(2,3,1); imshow(Req); title('Red Component Equalized','fontsize',14);
subplot(2,3,2); imshow(Geq); title('Green Component Equalized','fontsize',14);
subplot(2,3,3); imshow(Beq); title('Blue Component Equalized','fontsize',14);
subplot(2,3,4); imhist(Req); title('Red Component Equalized','fontsize',14); ylim([0 20000]);
subplot(2,3,5); imhist(Geq); title('Green Component Equalized','fontsize',14); ylim([0 70000]);
subplot(2,3,6); imhist(Beq); title('Blue Component Equalized','fontsize',14); ylim([0 63000]);
```











The original strawberry image had a higher density of RGB intensity values around the darker intensity lower limit. After the histogram equalization was processed, the distribution of the intensity values in the RGB images were spread across higher intensity values which resulted in the image becoming lighter because each RGB component became lighter as a result of the equalization.

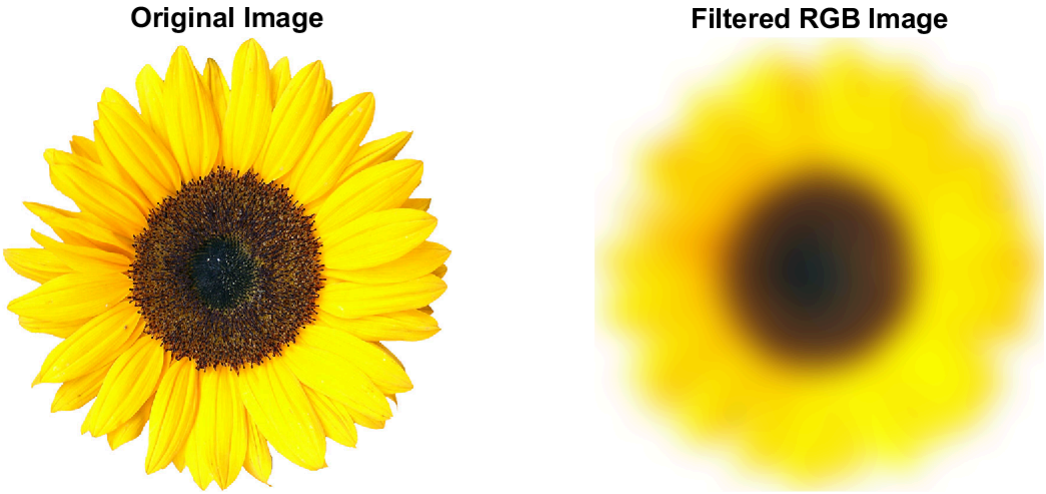
Problem-10 DIPUM3E Project 7.7 (Page 455)

Color Image Filtering

(a) Smoothing of `sunflower.tif` Image

Solution:

```
 f = imread('sunflower.tif');
 R = f(:,:,1);
 G = f(:,:,2);
 B = f(:,:,3);
 % Filter using a Gaussian Kernel that's large enough that resulting image
 % contains dark block in center
 sigma = 25;
 Rf = imgaussfilt(R,sigma);
 Gf = imgaussfilt(G,sigma);
 Bf = imgaussfilt(B,sigma);
 % Concatenate RGB images back into one image
 f_f = cat(3,Rf,Gf,Bf);
figure('Units','inches','Position',[0,0,7,3]);
subplot(1,2,1); imshow(f); title('Original Image','fontsize',14);
subplot(1,2,2); imshow(f_f); title('Filtered RGB Image','fontsize',14);pause(1)
```



(b) Sharpening of `sunflower.tif` Image

Solution:

```
% Create Laplacian Kernel based on Eq(3-29) and add 1 to center coefficient
% to account for HPF image
w1 = fspecial('laplacian',0);
w1(2,2) = w1(2,2) + 1;
% Create Laplacian Kernel based on Eq(3-30) and add 1 to center coefficient
% to account for HPF image
w2 = [1,1,1; 1,-8,1; 1,1,1];
w2(2,2) = w2(2,2) + 1;
% Filter original image with both Laplacian Kernels and discuss differences
% in results
f = im2double(f);
g1 = imfilter(f,w1,'replicate');
g2 = imfilter(f,w2,'replicate');
figure('Units','inches','Position',[0,0,7,3]);
subplot(1,2,1); imshow(g1,[]); title('Laplacian Kernel 1','fontsize',14);
subplot(1,2,2); imshow(g2,[]); title('Laplacian Kernel 2','fontsize',14);pause(1)
```

Laplacian Kernel 1

Laplacian Kernel 2

The reason why the 2nd Laplacian Kernel looks sharper is because the kernel used to generate it is an alternative kernel, compared to the first Laplacian kernel, and this kernel accounts for pixels in the diagonal direction when performing the convolution with the original image. This results in different pixels values compared to the first Laplacian kernel and has an overly sharpened look to it since it factored in more pixels during its operation.

```
function g = saltPepper4e(f,ps,pp)
%SALTPEPPER4E corrupts an image with salt-and-pepper noise.
% g = SALTPEPPER4E(F,PP,Ps) corrupts image F with salt-and-pepper
% noise. Pepper values are 0 with probability PP and salt va1ues
% are 1 with probability PS. Image F can be floating point or can
% have integers values. The output, corrupted image G, is floating
% point with values in the range [0,1]. This function is based on
% implementing Eq. (5-16) in DIP4E.
%

% Insert your code below.
% Convert image to floating-point between range of [0 1]
f = intScaling4e(f);
[M,N] = size(f);
if(ps + pp) > 1
    error('The Sum of PS and PP cannot exceed 1.')
end
X = rand(M,N);
R(1:M,1:N) = 0.5;
R(X <= pp) = 0;
R(X > pp & X <= ps + pp) = 1;
fn = f;
fn(R==1) = 1; % Salt
fn(R==0) = 0; % Pepper
fn = max(min(fn,1),0);
g = fn;
```

```
end

function L = constrainedLsTF4e(H,gam)
%CONSTRAINEDLSTF4e constrained least squares filter transfer function.
%  L = CONSTRAINEDLSTF4E(H,GAM) implements the constrained least
%  squares filter transfer function defined in Eq. (5-89) of DIP4E.
%  H is the transfer function of the degradation, and GAM is the
%  gamma parameter in Eq. (5-89). This function requires that H be
%  of even dimensions. If H was obtained from a degraded image
%  whose dimensions are not even, delete a row, column, or both, to
%  make the size of the image be even in both directions. Then recompute H.
% Copyright 2017, R. C. Gonzalez & R. E. Woods

% Insert your code below.
[M,N] = size(H);

if(iseven(M) ~= 1)
    H(M,:) = [];
    M = M - 1;
end
if(iseven(N) ~= 1)
    H(:,N) = [];
    N = N - 1;
end
X = zeros(M,N);
p = [0,-1,0;-1,4,-1;0,-1, 0];
X((M/2-1):(M/2+1),(N/2-1):(N/2+1)) = p;
P = abs(fft2(X)).^2;
Habs = abs(H).^2;
Hconj = conj(H);
L = (Hconj./(Habs + gam.*P));
end

function g = rgb2gray4e(f,method)
%RGB2GRAY4E Converts RGB image to grayscale.
% G = RGB2GRAY4E(F,METHOD) converts 24-bit RGB color image,
% F, to an 8-bit grayscale image, G. If METHOD = 'average',
% the average method is used. If METHOD = 'ntsc', the NTSC method
% is used. This is the defauit.

% Insert your code below
if nargin < 2
   method = 'ntsc';
end
f = im2double(f);
R = f(:,:,1);
G = f(:,:,2);
B = f(:,:,3);

switch method
    case 'average'
        g = (R + G + B)/3;
    case 'ntsc'
        g = 0.299*R + 0.587*G + 0.114*B;
```

```
end
end
```