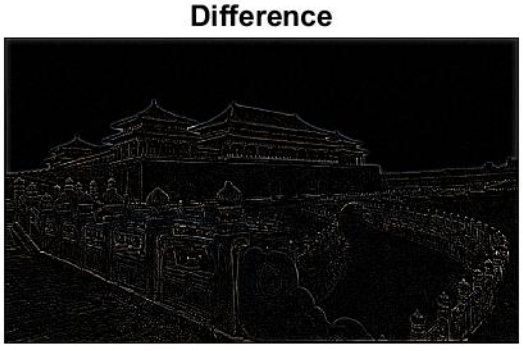
CS280 Homework 2

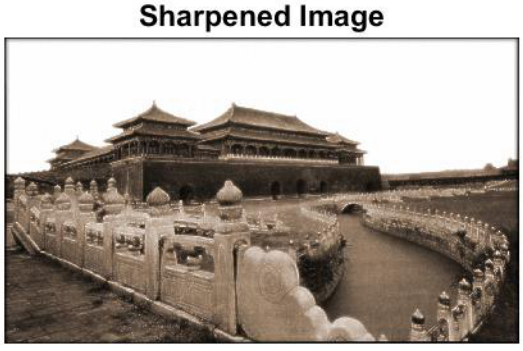
Jiaying Shi (SID: 24978491)

February 14, 2016

Problem 1:

I sharpened two photos. The results are shown in the following pictures.





Original Image

Blured Image

Difference

Sharpened Image

The procedure I used is to first blur the picture using Gaussian filter. And then use the original figure minus the blured figure to get the high frequency edges. Then the edges are scaled up and added back to the blured image to get the sharpened image. All results are shown in the above figures. The original pictures are found online and both are taken many years ago.

Code:

The code for problem 1 are listed as below:

y = gauss(x,std)

```
function y = gauss(x,std)
y = exp(-(x'*x)/(2*(std^2))) / (std^2*2*pi);
end
```

h = gaussFilter(n1,n2,std)

```
function h = gaussFilter(n1,n2,std)
for i = 1 : n2
    for j = 1 : n1
        u = [j-(n1+1)/2 ;i-(n2+1)/2];
        h(i,j) = gauss(u,std);
    end
end
h = h / sum(h(:));
end
```

problem1.m

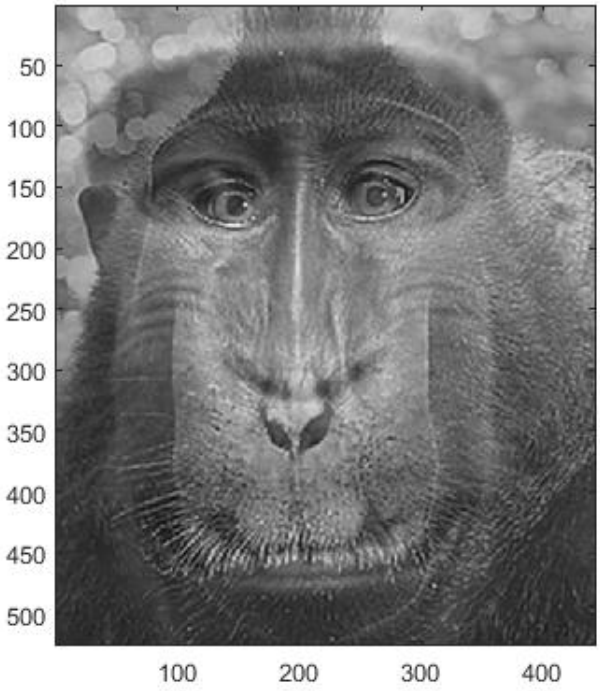
```
%%
%CS280 HW2 Problem 1
%Jiaying Shi
%shijy07@berkeley.edu
%%
clear all;
clc;
close all;
% IMG_FILE='photo.jpg';
IMG_FILE='tsinghua.jpg';
sigma=2;
hs=7;
hsize=[hs,hs];
gf=gaussFilter(hs,hs, sigma);

img=imread(IMG_FILE);
m=size(img,1);
n=size(img,2);

for c=1:3
    img_temp(:,:,c)=uint8(filter2(gf,img(:,:,c),'same'));
end
img_blur=img_temp;
img_diff=img-img_blur;
img_sharp=img_blur+2*img_diff;
figure;
subplot(2,2,1),imshow(img,[]),title('Original Image');
subplot(2,2,2), imshow(img_blur,[]),title('Blured Image');
subplot(2,2,3),imshow(img_diff*3,[]),title('Difference');
subplot(2,2,4), imshow(img_sharp,[]),title('Sharpened Image');
imwrite(img_blur,'blurttsinghua.jpg');
imwrite(img_sharp,'sharptsinghua.jpg');
imwrite(img_diff*4,'difftsinghua.jpg');
```

Other code used to generate the new images is in the submitted folder.

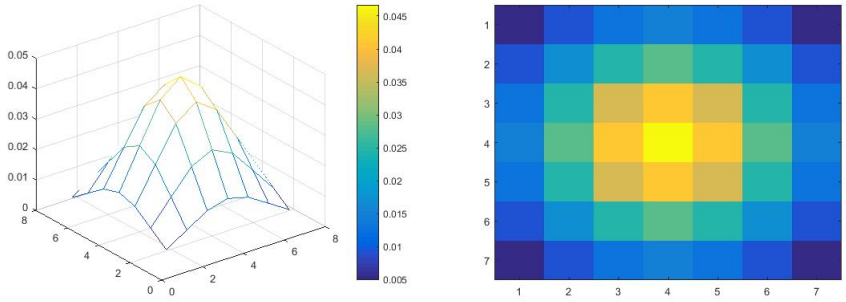
Problem 2:

In this algorithm, we use gaussian low pass filter and another high pass filter obtained from using a impulse filter minus the gaussian filter. After applying the low pass filter on one image and applying the high pass filter on the other image, and linearly adding the two images, we get the hybrided image. One of the hybrided images is from two apes with different expressions.The original images are shown as below:

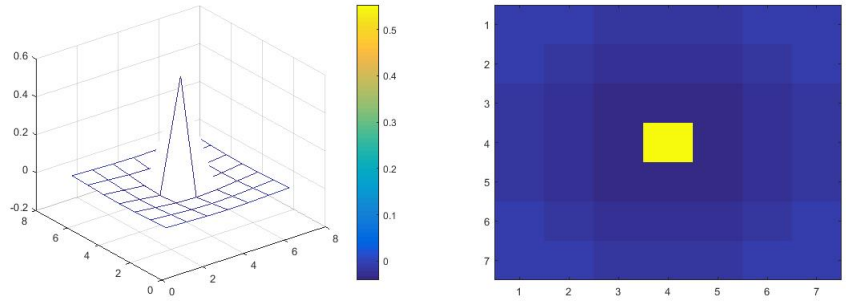
The hybrided image is shown as below:



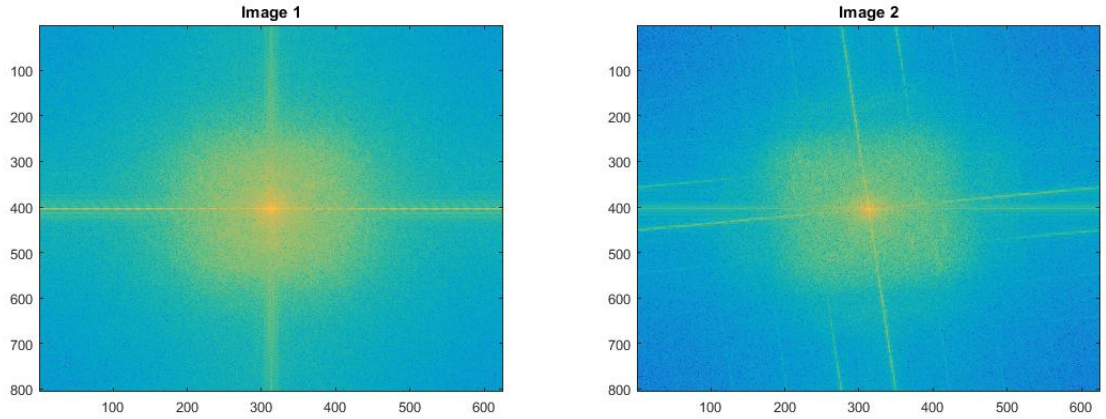
The gaussian filter is a 7×7 filter with $\sigma = 2$, the high pass filter is a impluse with magnitude 1 minus the previous gaussian filter. The spatial images of the low pass filter are shown as below:



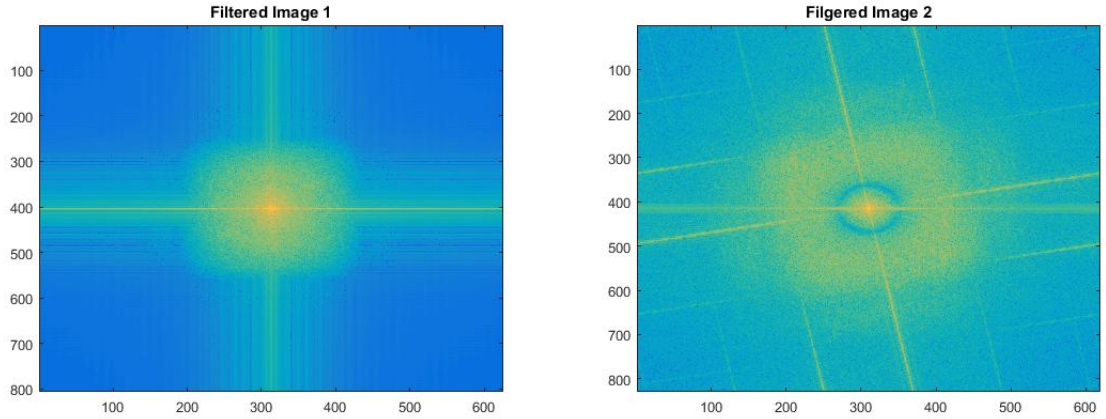
The spatial images of the high pass filter are shown as below:



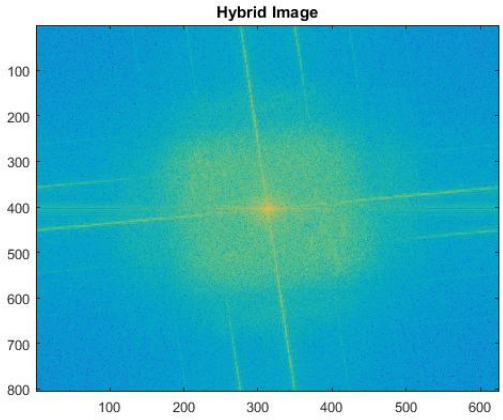
The 2 original images in frequency domain are shown in the following figures



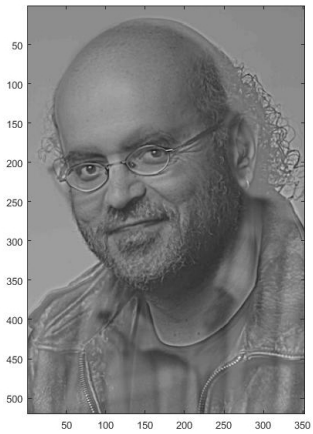
The filtered images in frequency domain are shown as below:

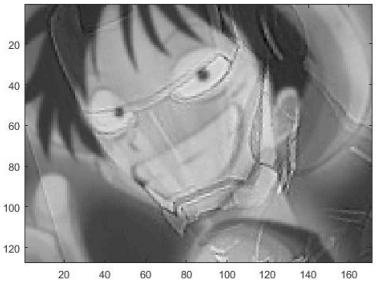


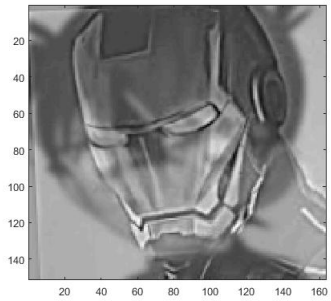
The hybrided image in frequency domain is shown as below:



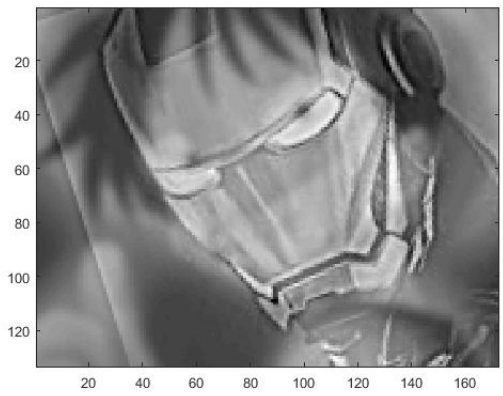
I also try this method on other images, the "not so bad" hybrided images are shown as below:







The fitst hybrid image has the best effect. If you zoom you can clearly see Professor Papadimitriou and if you zoom out down you will see Professor Malik.When I tried to hybrid Luffy and iron man I got a bad result as below:



The weight when combine the two image is not so good such that the Luffy image after filtered by low pass filter is too weak such that cannot be seen clearly.

Code:

The code for problem 1 are listed as below:

y = gauss(x,std)

```
function y = gauss(x,std)
y = exp(-(x'*x)/(2*(std^2))) / (std^2*2*pi);
end
```

h = gaussFilter(n1,n2,std)

```
function h = gaussFilter(n1,n2,std)
for i = 1 : n2
    for j = 1 : n1
        u = [j-(n1+1)/2 ;i-(n2+1)/2];
        h(i,j) = gauss(u,std);
    end
end
h = h / sum(h(:));
end
```

im12 = hybridImage(im1, im2, hs,sigma, alpha)

```
function im12 = hybridImage(im1, im2, hs,sigma, alpha)
H1=gaussFilter(hs,hs,sigma);
H2=zeros(hs,hs);
H2((hs+1)/2,(hs+1)/2)=alpha;
H2=H2-H1;
im1=filter2(H1,im1,'same');
im2=filter2(H2,im2,'same');
figure;
imagesc(log(abs(fftshift(fft2(im1)))));
title('Filtered Image 1');
figure;
imagesc(log(abs(fftshift(fft2(im2)))));
title('Filgered Image 2');
im12=(im1+2*im2);
im12=im12-min(im12(:));
im12=uint8(im12/max(im12(:))*255);

figure;
imagesc(log(abs(fftshift(fft2(im12)))));
title('Hybrid Image');
end
```

hybrid_image_starter.m

```
clear all;
close all; % closes all figures
% IM1_FILE='malik.png';
% IM2_FILE='papadimitriou.png';
% IM1_FILE='monkey2.jpg';
% IM2_FILE='monkey1.jpg';
IM1_FILE='conan.jpg';
IM2_FILE='ironman.jpg';
% read images and convert to single format
im1 = im2single(imread(IM1_FILE));
im2 = im2single(imread(IM2_FILE));
im1 = rgb2gray(im1); % convert to grayscale
im2 = rgb2gray(im2);
% use this if you want to align the two images (e.g., by the eyes) and crop
% them to be of same size
[im2a, im1a] = align_images(im2, im1);

% uncomment this when debugging hybridImage so that you don't have to keep aligning
% keyboard;
```

```
%% Choose the cutoff frequencies and compute the hybrid image (you supply
%% this code)
arbitrary_value = 100;
cutoff_low = arbitrary_value;
cutoff_high = arbitrary_value;
im12 = hybridImage(im1a, im2a, 5,2,0.8);

%% Crop resulting image (optional)
figure, hold off, imagesc(im12), axis image, colormap gray
disp('input crop points');
[x, y] = ginput(2);  x = round(x); y = round(y);
im12 = im12(min(y):max(y), min(x):max(x), :);

figure, hold off, imagesc(im12), axis image, colormap gray
figure;
imagesc(log(abs(fftshift(fft2(im1a)))));
title('Image 1');
figure;
imagesc(log(abs(fftshift(fft2(im2a)))));
title('Image 2');

%% Compute and display Gaussian and Laplacian Pyramids (you need to supply this function)
N = 3; % number of pyramid levels (you may use more or fewer, as needed)
A=pyrimid(im12, N);
figure;
imshow(A,[]);
```

Problem 3:

The problem can be written as

$$MatchScore_{u,v} = \sum_{h=0}^{H_1-1} \sum_{w=0}^{W_1-1} \left(T_{h,w} - I_{h+u,w+v}\right)^2$$

$$= \sum_{h=0}^{H_1-1} \sum_{w=0}^{W_1-1} \left(T_{h,w}^2 + I_{h+u,w+v}^2 - 2T_{h,w}I_{h+u,w+v}\right)$$

$$= trace\left(T^T T\right) + trace\left(\bar{I}_{u,v}^T \bar{I}_{u,v}\right) - 2\sum_{h=0}^{H_1-1} \sum_{w=0}^{W_1-1} T_{h,w}I_{h+u,w+v}$$

where $\bar{I}_{u,v}$ is the sub-matrix of I started from component u, v.
Let $\bar{T}$ be a matrix with

$$\bar{T}_{i,j} = T_{H_1-i,W_1-j}$$

From the above equation we can see that, the first term

$$trace\left(T^T T\right) = \left(\left(T \star \bar{T}\right) \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \right)_{u,v}$$

by using *conv2(T,Tbar,'valid')*ones(H2-H1,W2-W1)* in Matlab we can get this value for all components.
For the second term, we can first component wise take the square of the matrix I by using *Inew=I.^2* in Matlab.
Then, we can convolute the Inew a matrix with the same size of T and all components being one

$$trace\left(\bar{I}_{u,v}^T \bar{I}_{u,v}\right) = \left(Inew \star \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}_{H_1 \times W_1} \right)_{u,v}$$

by using *conv2(Inew,ones(H1,W1),'same')* we can get the matrix with proper dimension.
For the last term

$$m_{u,v} = 2\sum_{h=0}^{H_1-1} \sum_{w=0}^{W_1-1} T_{h,w}I_{h+u,w+v}$$

can also be interprated as an convolusion

$$m_{u,v} = \left(I \star T\right)_{u,v}$$

by using *conv2(I,Tbar,'same')* and then take the first $H_2 - H_1$ rows and $W_2 - W_1$ columns.
Thus, we can use three convolusions to get the matrix without looping ovre u and v.

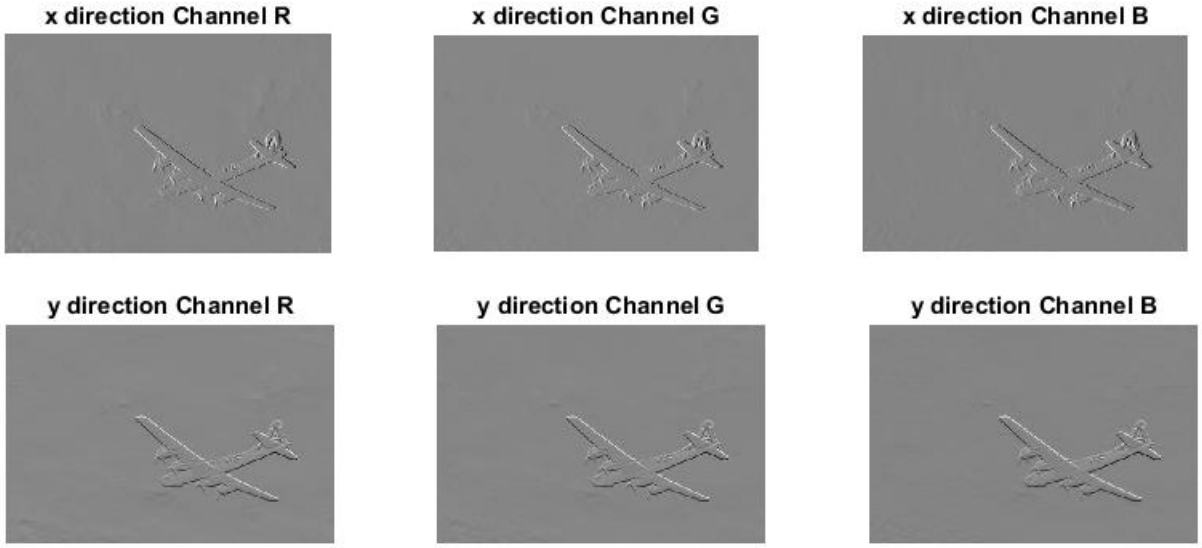
Problem 4:

1 Finite difference operator

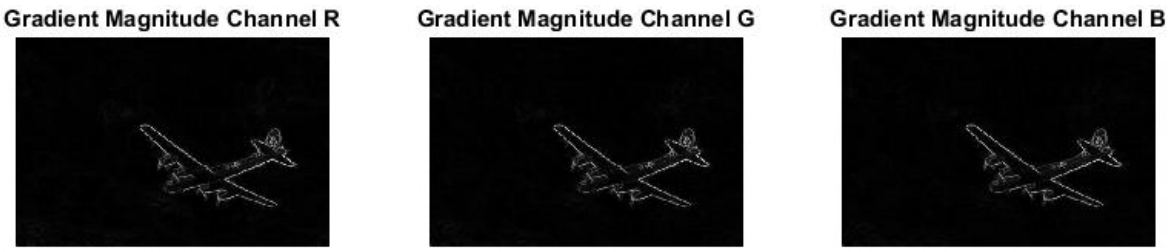
The convloutional filters D_x and D_y can be expressed as

$$D_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \; D_y = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Applying the finite difference operators on the image we get the images of the x and y direction difference of 3 channels as below:

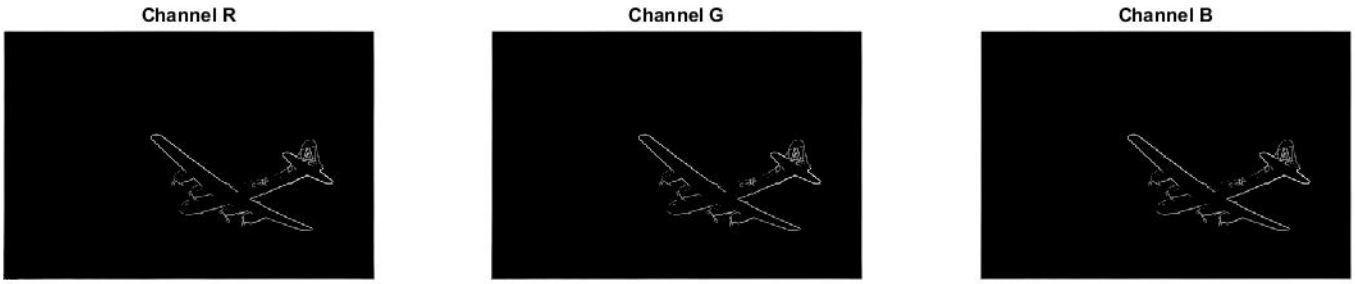


The magnitude of the difference on 3 channels are shown in the following figure:

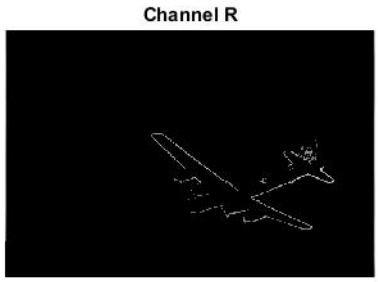


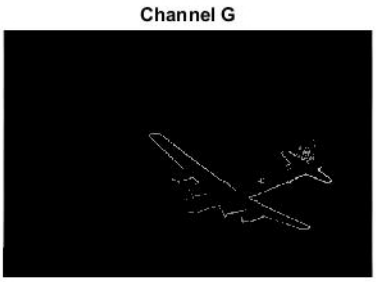
From the above figure we can see that the results are noisy. I used three threshold with respect to the maximum magnitude of the difference to denoise the magnitude.The results are shown as below:

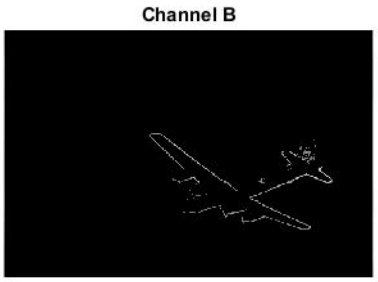
- $threshold = 0.2 * max(mag(:, :, c))$



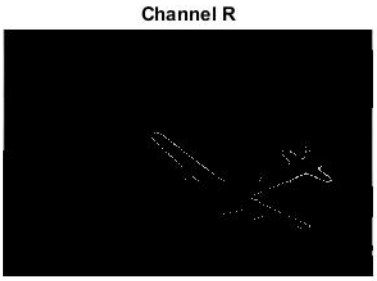
- $threshold = 0.4 * max(mag(:,:,c))$

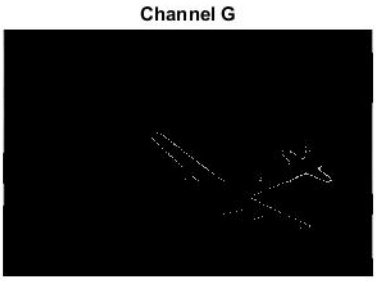


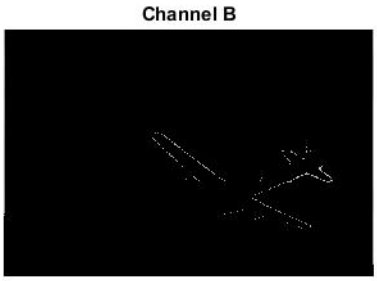




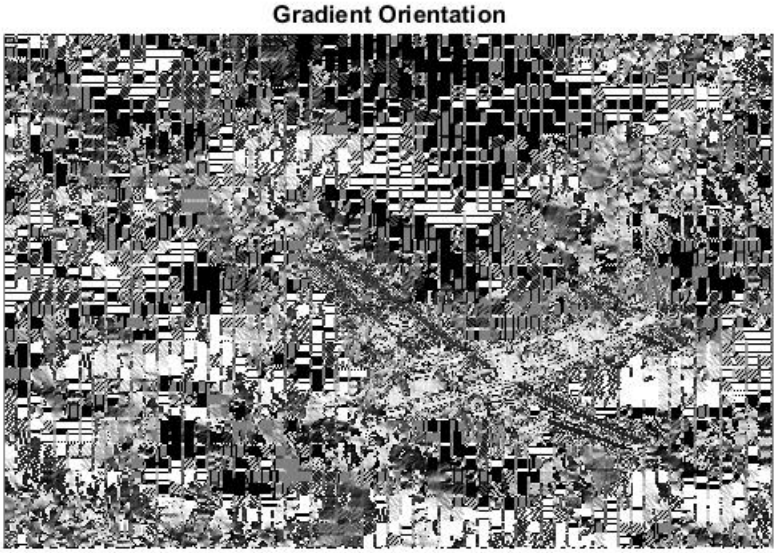
- $threshold = 0.6 * max(mag(:,:,c))$



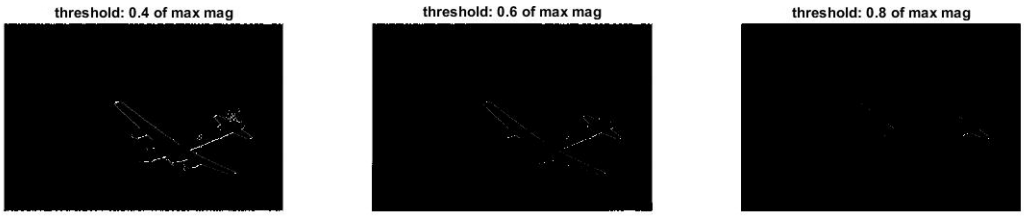




From the above results we can see that with proper threshold, the magnitude can be de-noised and the useful information is kept. But if the threshold value is too large, some parts of the edge will be missing.
The gradient orientation is calculated by using the directions of gradient corresponding to the channel with the largest gradient magnitude. Because we calculated the gradient in x and y direction which are just the projection of the actual gradient. Thus we can use arctan $\left(\frac{\partial f_y}{\partial f_x}\right)$ to calculate the theta. The orientations are depicted in the following figure:



Form the above picture we can see that the orientation is very noisy since we did not process the picture before calculating the direction of the gradient. Even if the magnitude of the gradient is small at some point, the angle can still be very large. For straight lines on the edge, the gradient direction value should be similar, which indicate a line with similar pixel value on the above picture. The angles can be pretty random for the background. In this case even if we set a threshold on theta, it will be meaning less. Thus I still threshold on the magnitude, the results are shown as below. From teh figure we can see that the points on the straight line has similar theta values.

d 

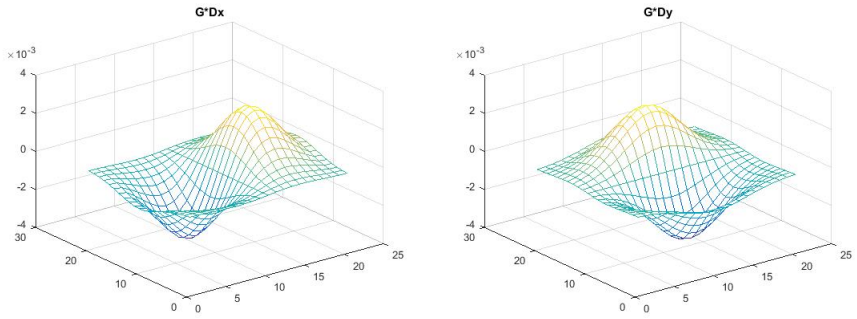
Code:

[mag, theta] = difference filter(img)

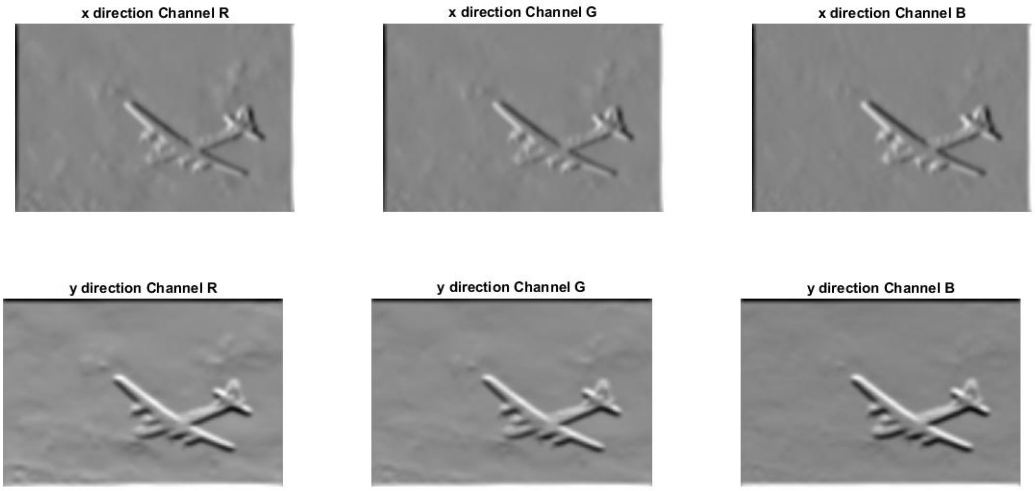
```
function [mag,theta] = difference_filter(I)
% compute magnitude and theta of image Ifunction[mag, theta]=difference_filter(img)
Dx=[0 0 0;-1 0 1;0 0 0];
Dy=[0 -1 0;0 0 0; 0 1 0];
nc=size(I,3);

for c=1:nc
    diffx(:,:,c)=conv2(double(I(:,:,c)),Dx,'same');
    diffy(:,:,c)=conv2(double(I(:,:,c)),Dy,'same');
    mag(:,:,c)=sqrt(diffx(:,:,c).^2+diffy(:,:,c).^2);
end
theta=zeros(size(I,1),size(I,2));

for i=1:size(I,1)
    for j=1:size(I,2)
        if max(mag(i,j,:))==mag(i,j,1)
            theta(i,j)=atan(diffy(i,j,1)/diffx(i,j,1));
        else
            if max(mag(i,j,:))==mag(i,j,2)
             theta(i,j)=atan(diffy(i,j,2)/diffx(i,j,2));
            else
                theta(i,j)=atan(diffy(i,j,3)/diffx(i,j,3));
            end
        end
    end
end
for i=1:nc
    mag(:,:,i)=uint8((mag(:,:,i)-min(min(mag(:,:,i))))...
        /(max(max(mag(:,:,i)))-min(min(mag(:,:,i))))*255);
end

theta=theta-min(theta(:));

figure;
subplot(1,3,1), imshow(diffx(:,:,1),[]); title('x direction Channel R');
subplot(1,3,2), imshow(diffx(:,:,2),[]); title('x direction Channel G');
subplot(1,3,3), imshow(diffx(:,:,3),[]); title('x direction Channel B');
figure;
subplot(1,3,1), imshow(diffy(:,:,1),[]); title('y direction Channel R');
subplot(1,3,2), imshow(diffy(:,:,2),[]); title('y direction Channel G');
subplot(1,3,3), imshow(diffy(:,:,3),[]); title('y direction Channel B');
```

2 Derivative of Gaussian

The Gaussian filter I used for this problem is of dimension 15 and the standard deviation is 3. The filters $G_\sigma \star D_x$ and $G_\sigma \star D_y$ are shown as below:



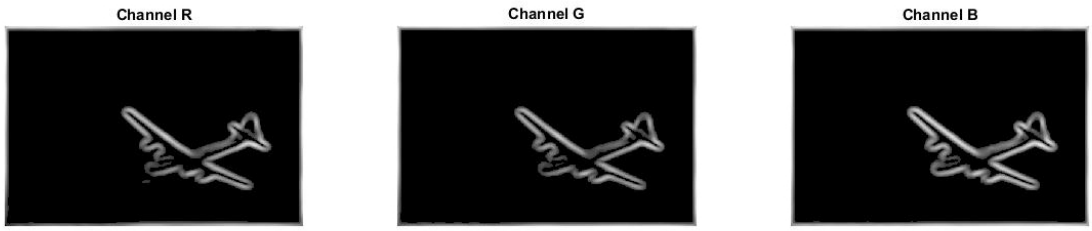
Applying the finite difference operators on the image we get the images of the x and y direction difference of 3 channels as below:



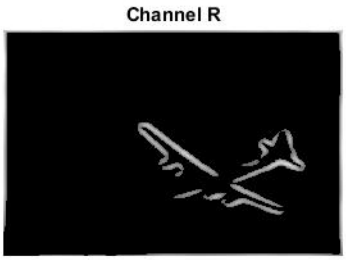
The magnitude of the difference on 3 channels are shown in the following figure:

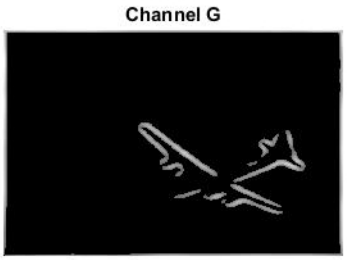
From the above figure we can see that the results are noisy. I used three threshold with respect to the maximum magnitude of the difference to denoise the magnitude.The results are shown as below:

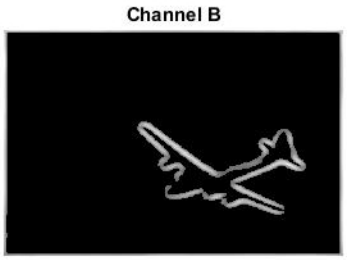
- $threshold = 0.2 * max(mag(:, :, c))$



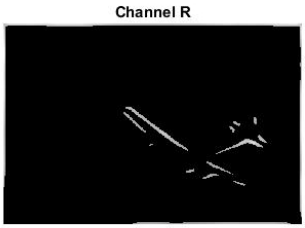
- $threshold = 0.4 * max(mag(:, :, c))$

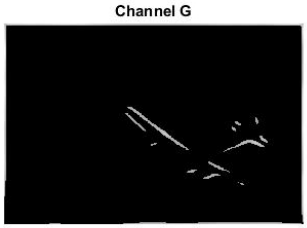


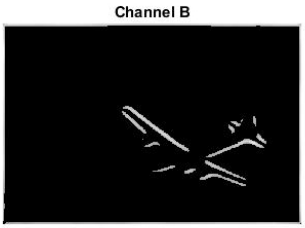




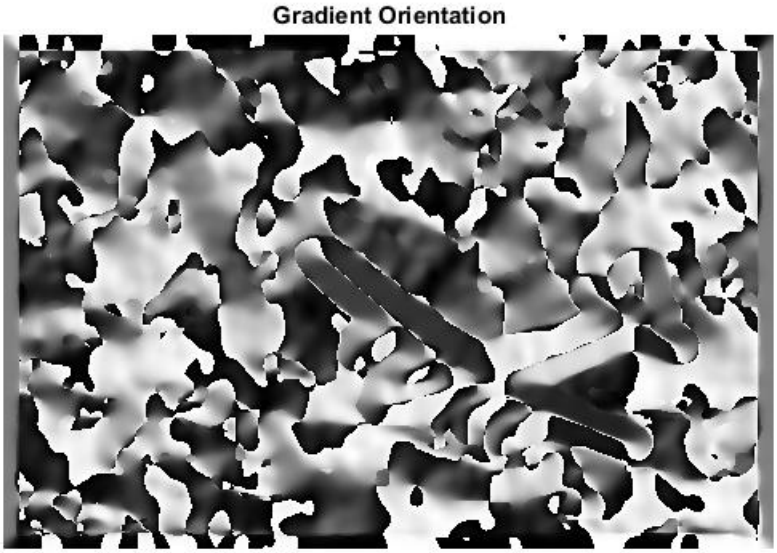
- $threshold = 0.6 * max(mag(:,:,c))$



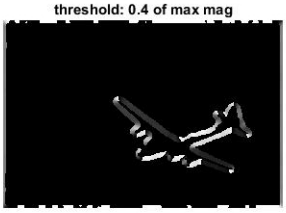


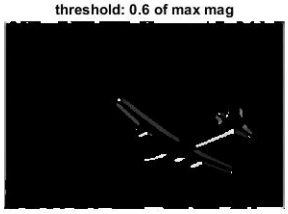


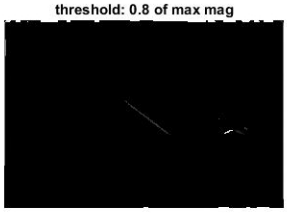
The gradient orientations which are calculated in the same way as in part 1 are depicted in the following figure:



Similar as in the previous, from the threshold figure we can see that on the edge, parallel lines has similar theta values. And at the cornors, the theta values changes alot.







Observation:

By applying Gaussian filter, we avoid the noise from just using the difference filters. The edges obtained in the second part are thicker than those in the first part. The edges changes more smooth. When I try to threshold using different values, we got similar results. When the threshold value is too large, we will loss some of the edges. For the orientation image, when using derivative of Gaussian there is much less noise. The angle of the gradient changes more smoothly.

Code:

[mag,theta] = derivative_gaussian_filter(I, sigma,hs)

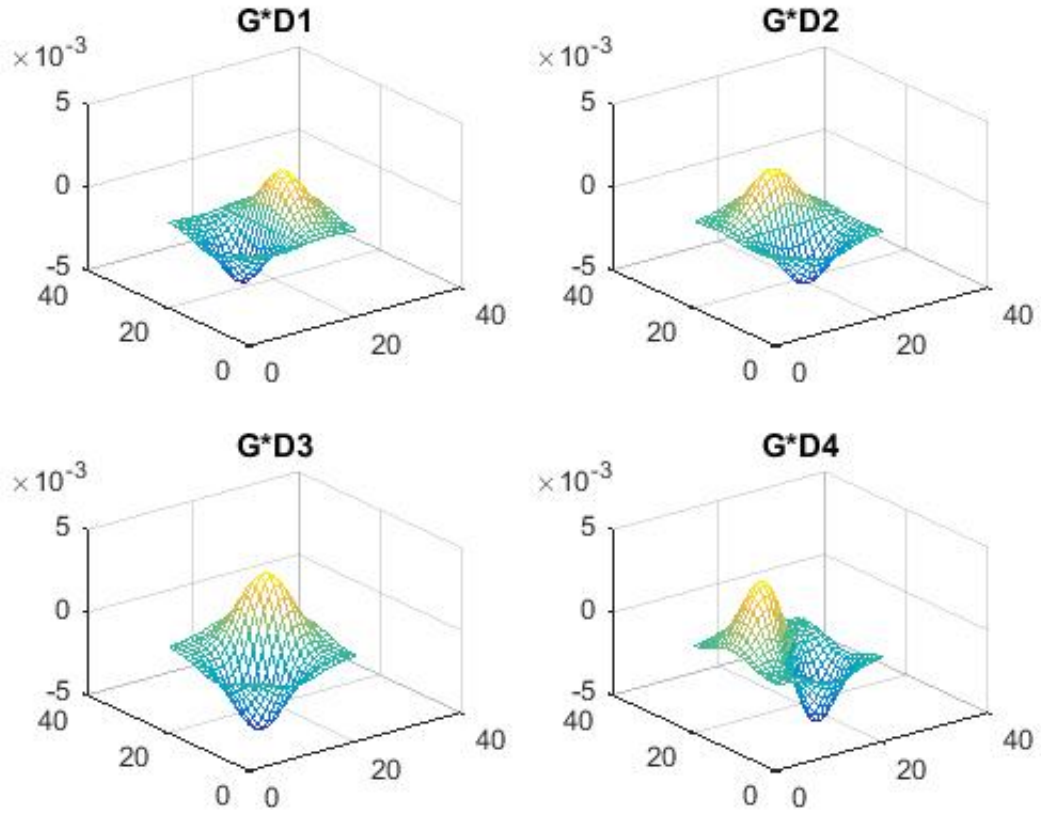
```
function [mag,theta] = derivative_gaussian_filter(I, sigma,hs)
% compute magnitude and theta of image I
Dx=[0 0 0;-1 0 1;0 0 0];
Dy=[0 -1 0;0 0 0; 0 1 0];
nc=size(I,3);
Hg=gaussFilter(hs,hs,sigma);
Hx=conv2(Hg,Dx,'same');
Hy=conv2(Hg,Dy,'same');
figure;
mesh(Hx);title('G*Dx')
figure;
mesh(Hy); title('G*Dy');
nc=size(I,3);



for c=1:nc
    diffx(:,:,c)=conv2(double(I(:,:,c)),Hx,'same');
    diffy(:,:,c)=conv2(double(I(:,:,c)),Hy,'same');
    mag(:,:,c)=sqrt(diffx(:,:,c).^2+diffy(:,:,c).^2);
end
theta=zeros(size(I,1),size(I,2));

for i=1:size(I,1)
    for j=1:size(I,2)
        if max(mag(i,j,:))==mag(i,j,1)
            theta(i,j)=atan(diffy(i,j,1)/diffx(i,j,1));
        else
            if max(mag(i,j,:))==mag(i,j,2)
             theta(i,j)=atan(diffy(i,j,2)/diffx(i,j,2));
            else
                theta(i,j)=atan(diffy(i,j,3)/diffx(i,j,3));
            end
        end
    end
end
for i=1:nc
    mag(:,:,i)=uint8((mag(:,:,i)-min(min(mag(:,:,i))))...
        /(max(max(mag(:,:,i)))-min(min(mag(:,:,i))))*255);
end

theta=theta-min(theta(:));

figure;
subplot(1,3,1), imshow(diffx(:,:,1),[]); title('x direction Channel R');
subplot(1,3,2), imshow(diffx(:,:,2),[]); title('x direction Channel G');
subplot(1,3,3), imshow(diffx(:,:,3),[]); title('x direction Channel B');
figure;
subplot(1,3,1), imshow(diffy(:,:,1),[]); title('y direction Channel R');
subplot(1,3,2), imshow(diffy(:,:,2),[]); title('y direction Channel G');
subplot(1,3,3), imshow(diffy(:,:,3),[]); title('y direction Channel B');
```

3 Oriented Filters.

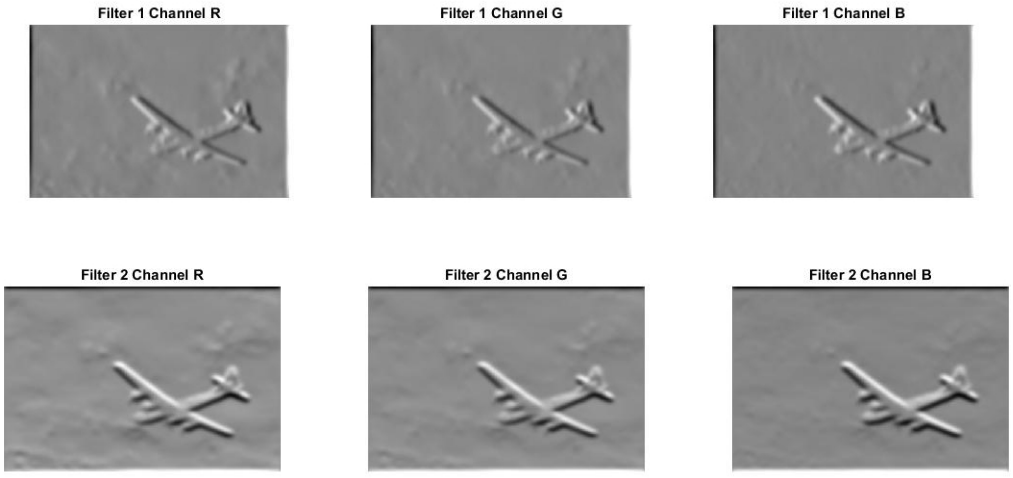
In this part, we use 4 filters

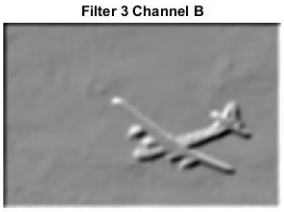
$$D_1 = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, D_1 = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, D_1 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, D_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

The filters are shown as below:



Applying the finite difference operators on the image we get the images of the x and y direction difference of 3 channels as below:

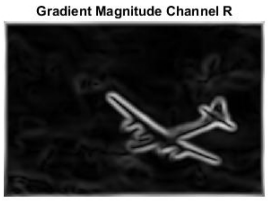


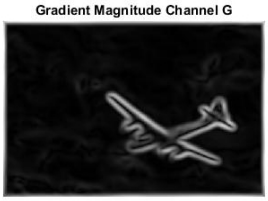


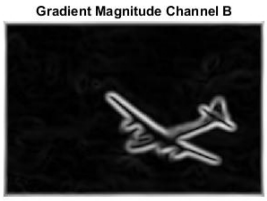
When we apply the above four filters on the image, to calculate the magnitude of gradient on point (i, j, c) , I use the following formula

$$mag_{i,j,c} = max\left(\sqrt{(G_\sigma \star D_1)^2_{i,j,c} + (G_\sigma \star D_1)^2_{i,j,c}}, \sqrt{(G_\sigma \star D_3)^2_{i,j,c} + (G_\sigma \star D_4)^2_{i,j,c}}\right)$$

The reason we use the above formula is because for a 2d space, 2 directions are enough to calculate the magnitude. But for images, the gradient is discreted and take the maximum of two values may strengthen the value of gradient. The magnitude of the difference on 3 channels are shown in the following figure:

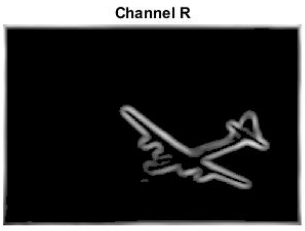


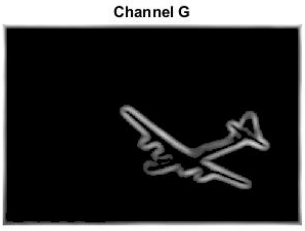


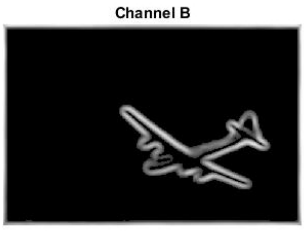


The above magnitude of gradient is stronger than the previous case using only 2 . I used three threshold with respect to the maximum magnitude of the difference to denoise the magnitude.The results are shown as below:

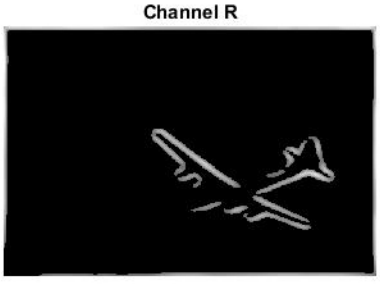
- $threshold = 0.2 * max(mag(:, :, c))$

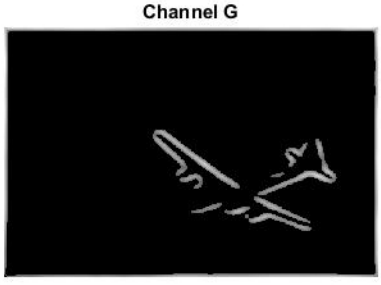


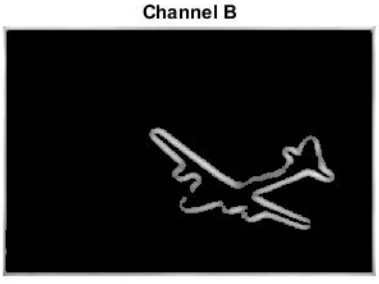




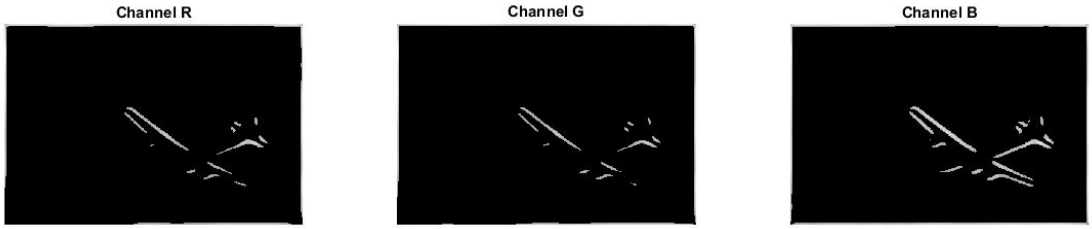
- $threshold = 0.4 * max(mag(:, :, c))$



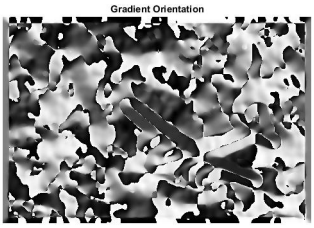




- $threshold = 0.6 * max(mag(:,:,c))$



The gradient orientations which are calculated in the same way as in part 1 are depicted in the following figure:



Similar as in the previous, from the threshold figure we can see that on the edge, parallel lines has similar theta values. And at the cornors, the theta values changes alot.

The human annotation is pretty different from the results obtained using pointwise filtering methods. When human try to find boundaries, they will use there persnoal knowledge. They will ignore some of the edges which is from the change of the texure or created by shadow. However for computer programs, they can only tell from the difference from the values of the pixels. Thus they can not tell the difference between actual edges and other types of pixel value changes.

Code:

[mag,theta] = oriented_filter(I,sigma,hs)

```
function [mag,theta] = oriented_filter(I,sigma,hs)
% compute magnitude and theta of image I
D1=[0 0 0;-1 0 1;0 0 0];
D2=[0 -1 0;0 0 0;0 1 0];
D3=[-1 0 0;0 0 0;0 0 1];
D4=[0 0 -1;0 0 0;1 0 0];
Hg=gaussFilter(hs,hs,sigma);
H1=conv2(Hg,D1,'same');
H2=conv2(Hg,D2,'same');
H3=conv2(Hg,D3,'same');
H4=conv2(Hg,D4,'same');
figure;
subplot(2,2,1), mesh(H1);title('G*D1')
subplot(2,2,2), mesh(H2);title('G*D2');
subplot(2,2,3), mesh(H3);title('G*D3')
subplot(2,2,4), mesh(H4); title('G*D4');
```

```
nc=size(I,3);
for c=1:nc
    diff1(:,:,c)=conv2(double(I(:,:,c)),H1,'same');
    diff2(:,:,c)=conv2(double(I(:,:,c)),H2,'same');
    diff3(:,:,c)=conv2(double(I(:,:,c)),H3,'same');
    diff4(:,:,c)=conv2(double(I(:,:,c)),H4,'same');
    mag(:,:,c)=sqrt(4*diff1(:,:,c).^2+4*diff2(:,:,c).^2+diff3(:,:,c).^2+diff4(:,:,c).^2);
end
theta=zeros(size(I,1),size(I,2));

for i=1:size(I,1)
    for j=1:size(I,2)
        if max(mag(i,j,:))==mag(i,j,1)
            theta(i,j)=atan(diff3(i,j,1)/diff4(i,j,1));
        else
            if max(mag(i,j,:))==mag(i,j,2)
             theta(i,j)=atan(diff3(i,j,2)/diff4(i,j,2));
            else
                theta(i,j)=atan(diff3(i,j,3)/diff4(i,j,3));
            end
        end
    end
end
for i=1:nc
    mag(:,:,i)=uint8((mag(:,:,i)-min(min(mag(:,:,i))))...
        /(max(max(mag(:,:,i)))-min(min(mag(:,:,i))))*255);
end

theta=single((theta-min(theta(:)))/pi*180);

figure;
subplot(1,3,1), imshow(diff1(:,:,1),[]); title('Filter 1 Channel R');
subplot(1,3,2), imshow(diff1(:,:,2),[]); title('Filter 1 Channel G');
subplot(1,3,3), imshow(diff1(:,:,3),[]); title('Filter 1 Channel B');
figure;
subplot(1,3,1), imshow(diff2(:,:,1),[]); title('Filter 2 Channel R');
subplot(1,3,2), imshow(diff2(:,:,2),[]); title('Filter 2 Channel G');
subplot(1,3,3), imshow(diff2(:,:,3),[]); title('Filter 2 Channel B');

figure;
subplot(1,3,1), imshow(diff3(:,:,1),[]); title('Filter 3 Channel R');
subplot(1,3,2), imshow(diff3(:,:,2),[]); title('Filter 3 Channel G');
subplot(1,3,3), imshow(diff3(:,:,3),[]); title('Filter 3 Channel B');
figure;
subplot(1,3,1), imshow(diff4(:,:,1),[]); title('Filter 4 Channel R');
subplot(1,3,2), imshow(diff4(:,:,2),[]); title('Filter 4 Channel G');
subplot(1,3,3), imshow(diff4(:,:,3),[]); title('Filter 4 Channel B');
end
```

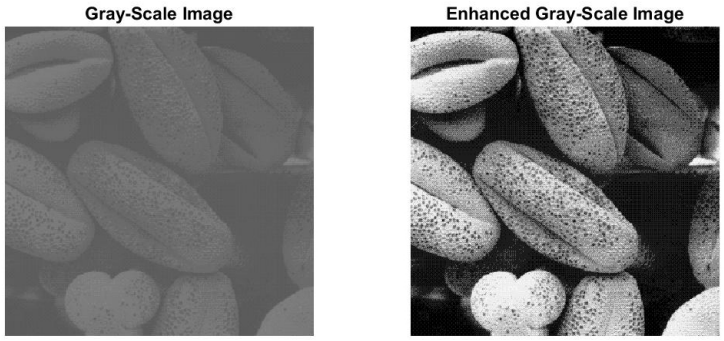
Problem 5

Part 1

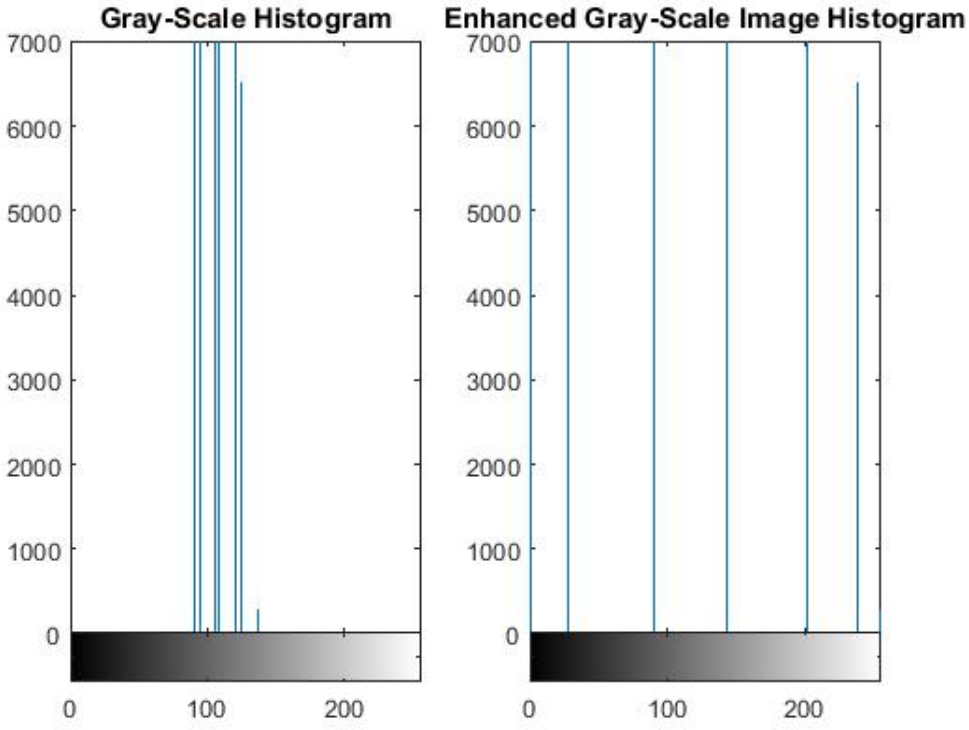
To get the new image that is closest to input and with uniform histogram, we can use the following procedure:

$$u_{ij} = floor(255 \sum_{n=0}^{t_{ij}} f_n^t)$$

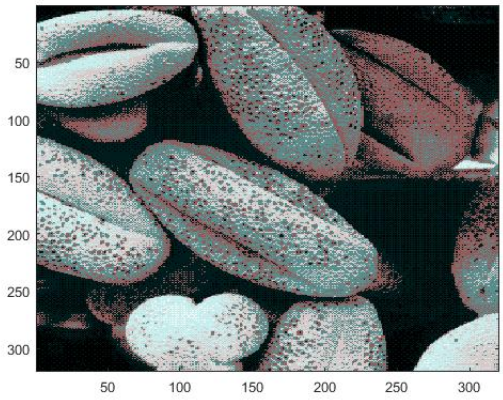
where f_n^t is the number of pixels with intensity n devided by the total number of pixels. In the above we are acturally applying the the cumulative distribution function of the original picture on it's pixel value to get the new image such that the histogram of the new image is uniform.
The gray scale results of the problem is shown as below:



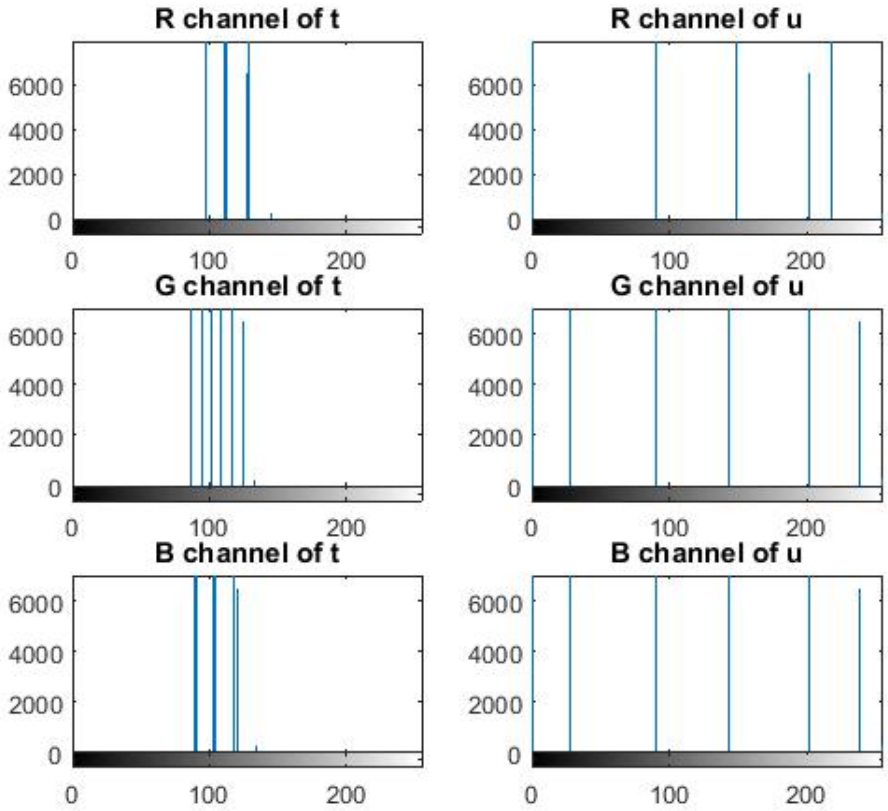
The histogram of the above two images are shown as below:



The results of the color image are shown as below:



The histogram of the original and enhanced images in three channels are shown as below:



Code:

u = histogram_equalize(t)

```
function u = histogram_equalize(t)
% Minimally change values of t so that the result has a uniform distribution
N=imhist(t);
N=N/sum(N);
L=255;
m=size(t,1);
n=size(t,2);
```

```
for i=1:m
    for j=1:n
        value=0;
        for k=1:t(i,j)
            value=value+N(k);
        end
        u(i,j)=uint8(floor(255*value));
    end
end
```

Part 2

To implement the histogram transfer function, we can match the cumulative frequency of two images. For example if we know that for pixel value x on image t the frequency is f_x^t. We want the pixel value on image value to be y where $f_y^s = f_x^t$.
The code is attached as below:

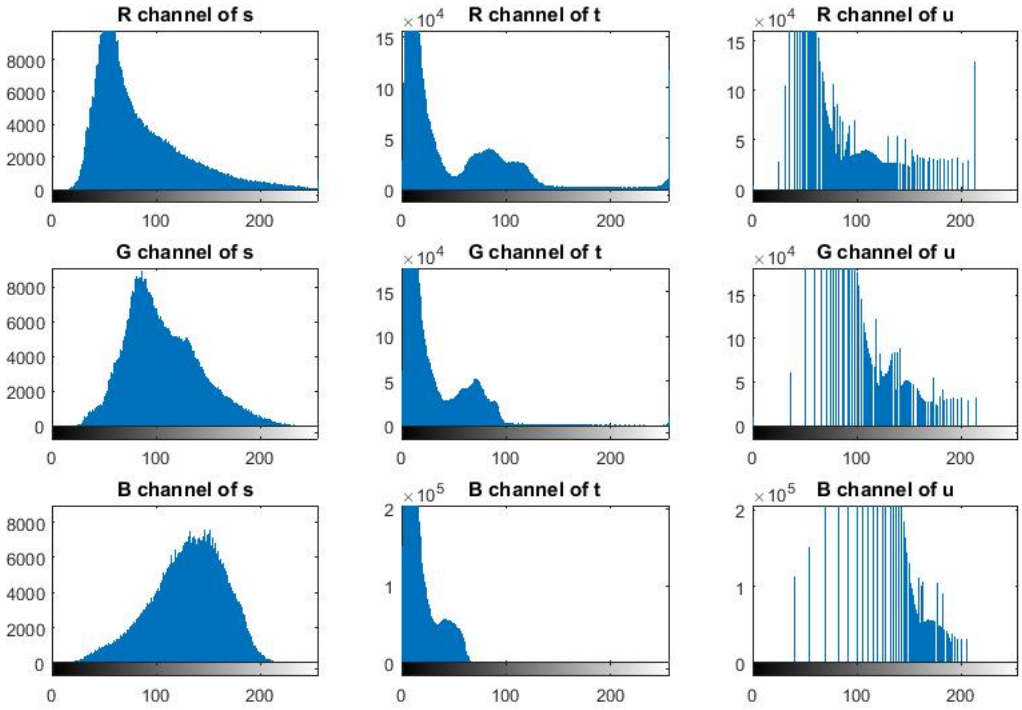
```
function u = histogram_transfer(s,t)
% Minimally change values of t so that they have the same statistics as s
hs=imhist(s);
hs=hs/sum(hs);
ht=imhist(t);
ht=ht/sum(ht);
cums=cumsum(hs);
cums=floor(255*cums);
cumt=cumsum(ht);
cumt=floor(255*cumt);
m=size(t,1);
n=size(t,2);
hu=hs*m*n;

for i=1:m
    for j=1:n
        probt=cumt(t(i,j)+1);
        temp=abs(cums-probt);
        [minv,indx]=min(temp);
        u(i,j)=uint8(indx-1);
    end
end
```

Part 3

The three images are attached as below:



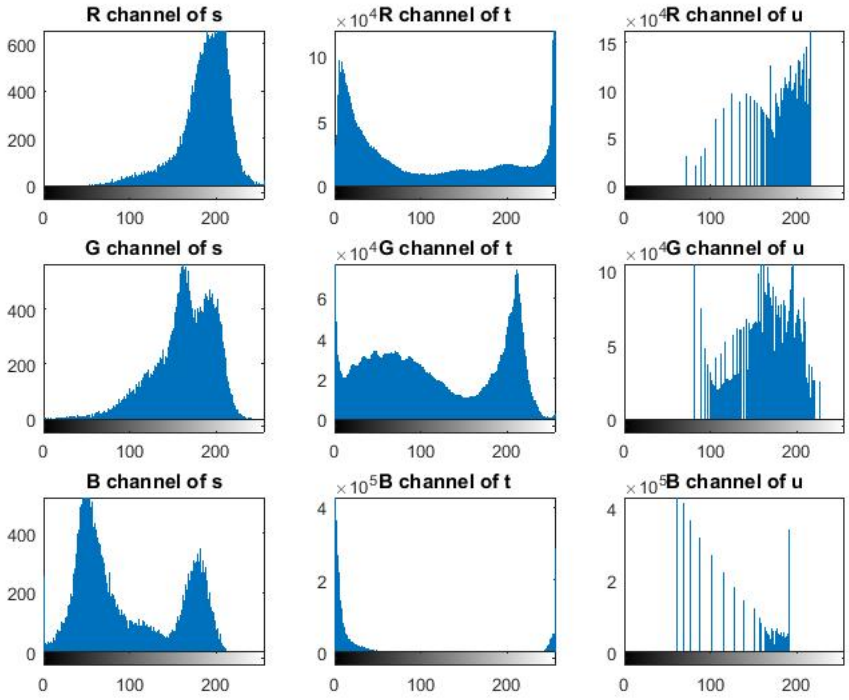
The histograms of three channels are shown as below:



Part 4:

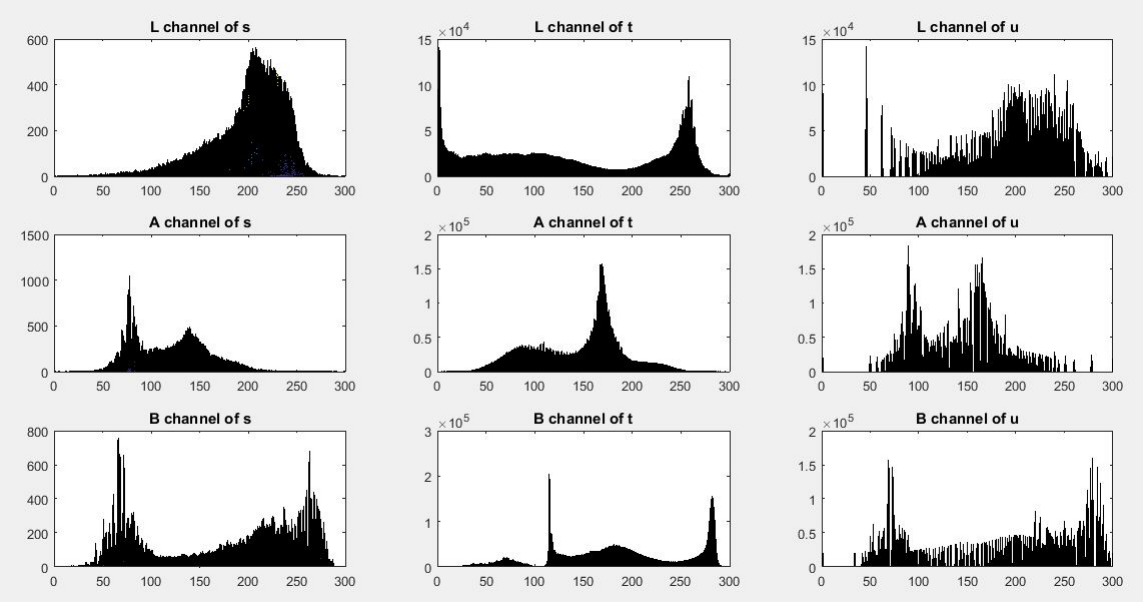
The results for my own images are shown as below:

The histograms of three channels are shown as below:



When we transfer the image to LAB first and then apply the histogram matching, the results are shown as below:

The histograms of three channels are shown as below:



The code is attached as below:

```
function u = histogram_transferLAB(s,t,L)
% Minimally change values of t so that they have the same statistics as s

hs=hist(s,L);
hs=hs/sum(hs);
ht=hist(t,L);
ht=ht/sum(ht);
cums=cumsum(hs);
cums=floor(L*cums);
cumt=cumsum(ht);
cumt=floor(L*cumt);
steps=(max(s(:))-min(s(:)))/(L-1);
stept=(max(t(:))-min(t(:)))/(L-1);
mint=min(t(:));
mins=min(s(:));
m=size(t,1);
n=size(t,2);
for i=1:m
    for j=1:n
        probt=cumt(floor((t(i,j)-mint)/stept)+1);
        temp=abs(cums-probt);
        [minv,indx]=min(temp);
        u(i,j)=steps*(indx-1)+mins;
    end
end
```

Part 5

Comparing with RGB color format, the LAB color format is designed to approximate human vision. It aspires to perceptual uniformity, and its L component closely matches human perception of lightness. If we transfer the image in LAB space, more information is contained in the image, thus the results is better.