ECE 253: HW3

Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and

in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind.

By including this in my report, I agree to abide by the Academic Integrity Policy mentioned above.

****

To run all files at once, load folder named “main” into Current Folder section in MATLAB(This contains all scripts as well as all images used). Then run “main” in the MATLAB console. This file will output results for the whole HW. To run each section individually, follow directions in each section below.

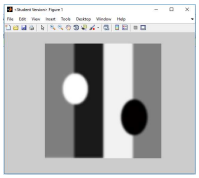
****

Ajinkya Bagde
U07800884
11/18/2017

Problem 1: Blurring and Sharpening

To run:
prob1main.m

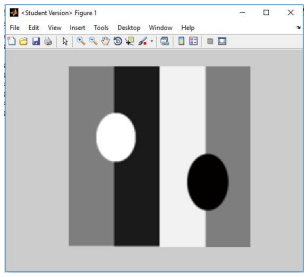
i) Implemented gaussianBlur() function. Image below is with sigma=2



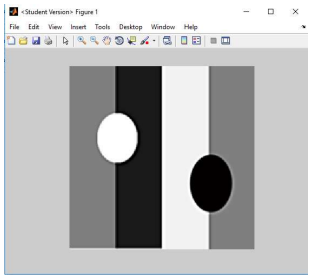
ii) Implemented blurOrSharpen() function

a. w = 1 gives the most sharpening, w = -1 gives the most blurring. For w = -1, the equation becomes I' = I*G_sigma, which is the same equation implemented in the guassianBlur() function in part 1. For any values between -1 and 0, the image will be blurred, and for any values between 0 and 1, the image will be sharpened. Looking at the equation, we can see that the weight just controls the scaling of the mask that is added to the original image. Images below are with sigma=2

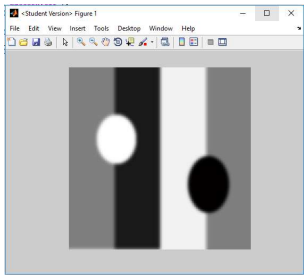
   Original image:



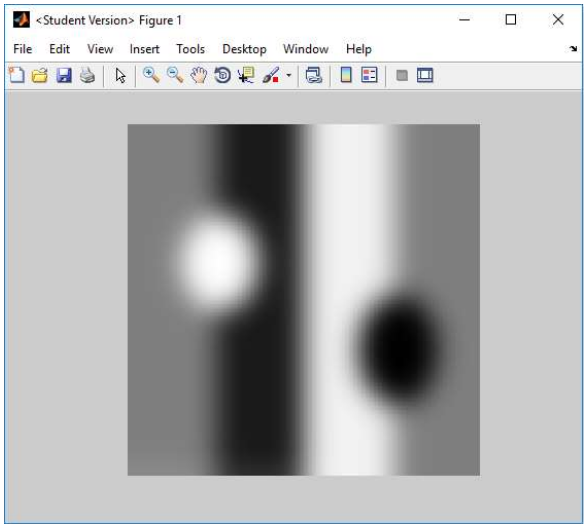
   W = 1:



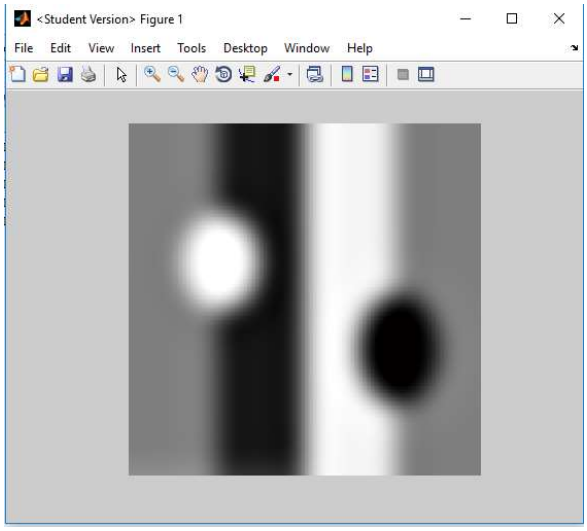
   W = -1:



b. Image blurred with sigma=10, w = -1:

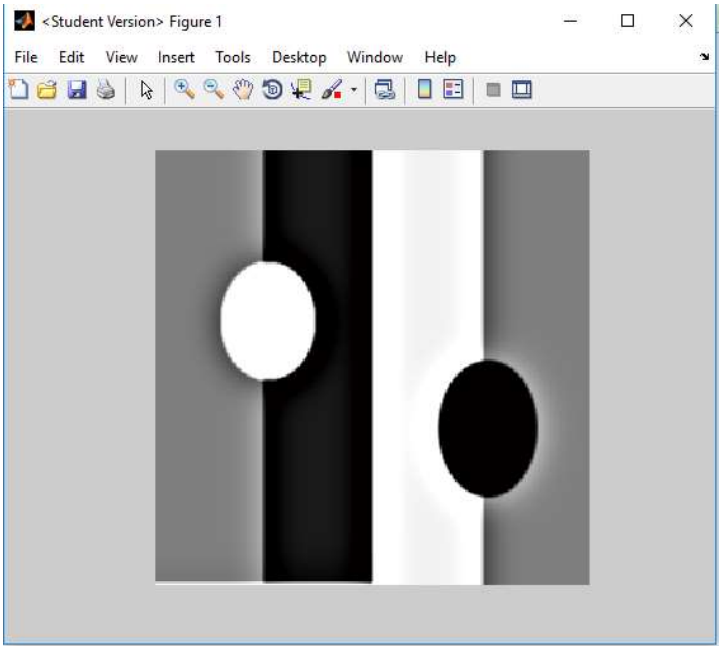


Now the above image sharpened with sigma=10, w = 1:



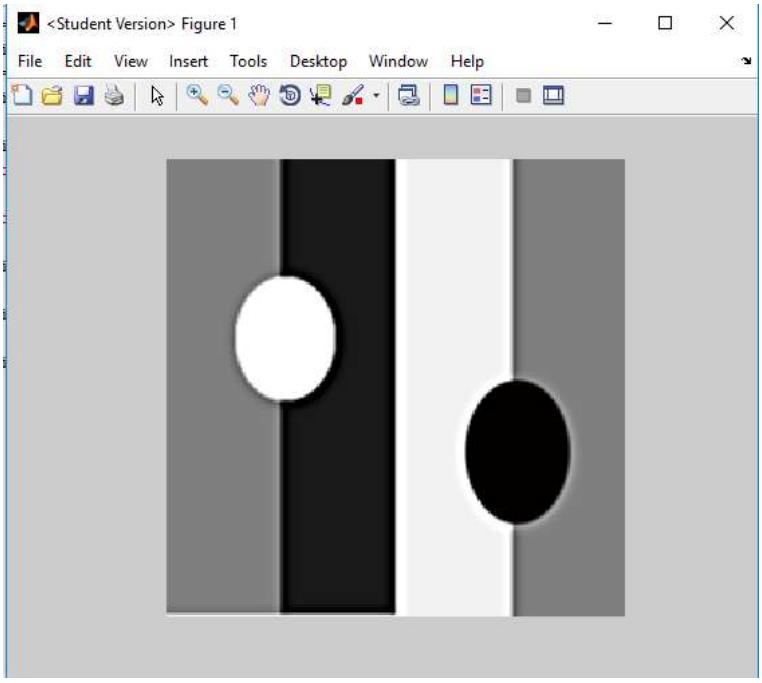
There is certainly improvement in the sharpened image, but we are nowhere close to the sharpness of the original image. Again, when blurring with w = -1, I' = I*G_sigma. If we now substitute I' back as I into equation (1) as specified in the problem we get something like this: (1+w) x (I*G_sigma) - w x (I*G_sigma). This does not simplify to I, so we should not expect to get the original image back.

c. Original image sharpened(w=1,sigma=10):



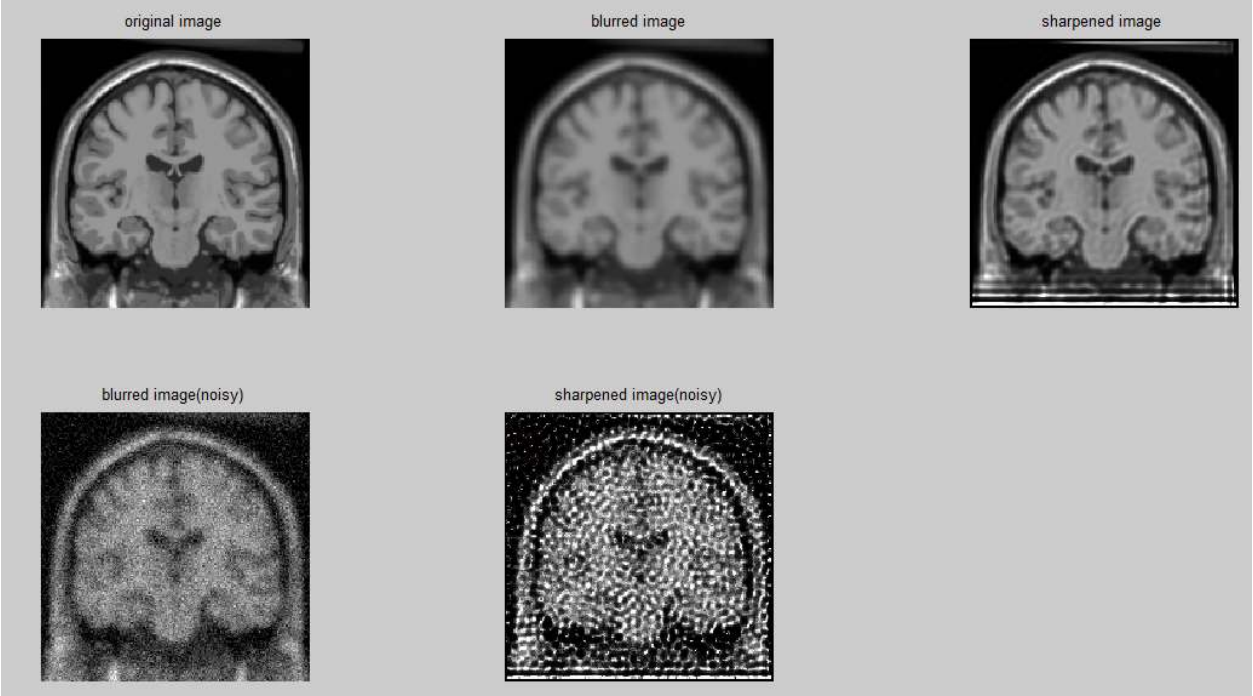
We can definitely observe that there are generated artifacts around the ovals. Referencing the following link http://www.cambridgeincolour.com/tutorials/unsharp-mask.htm, this has occured as a result of “too much sharpening”, especially with sigma = 10. Looking at the image with a lower radius value for the mask, the artifacts do not seem so pronounced:

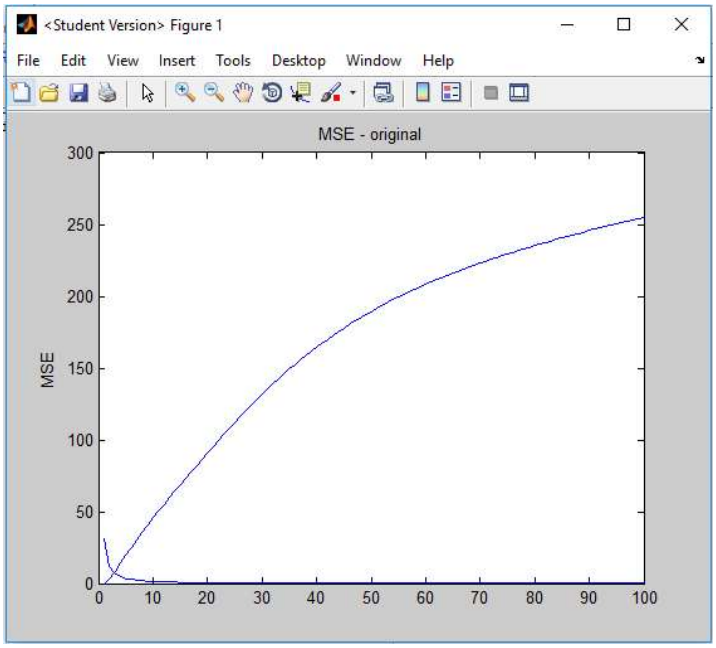
Original image sharpened(w=1,sigma=5):



iii)

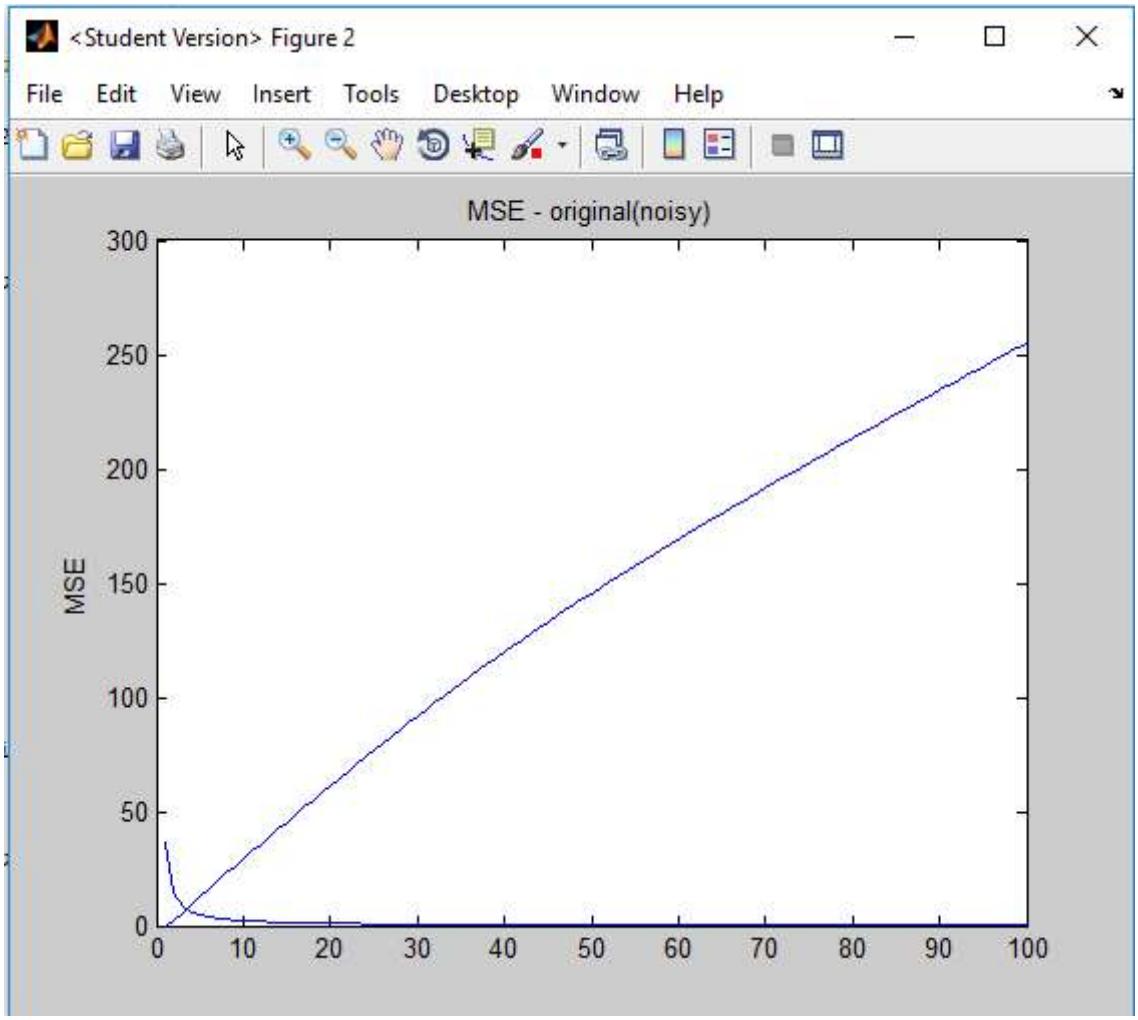
a.





The bottom plot is the MSE of residuals, the top curve is the MSE between the filtered and original image. The filtered image looks pretty similar to the original image, with the exception of the white line artifacts generated at the bottom. Keep in mind, these were generated after 100 iterations, and MSE between residuals was less than 0.0001. I realize my MSE plot between the original image and filtered image may not be correct(one would expect it to decrease, and eventually converge on a reasonable value, mine seems to be increasing at a slower rate and converging). This may be because of the artifacts generated.

b.



The bottom plot is the MSE of residuals, the top curve is the MSE between the filtered and original image. Actual images are in the subplot in part a. This MSE curve makes sense. The noise is visibly being amplified during the filtering, which correlates to the curve above.

```
%Problem 1: Blurring and Sharpening

clear;
clc;

%i)Using the guassianBlur() function
img = imread('brain.tif');
img = single(img);

blur_image = gaussianBlur(img,2);
%imshow(uint8(blur_image))

%ii)Using the blurOrSharpen() function
blur_sharpen_img = blurOrSharpen(img,2,1);
%figure
%imshow(uint8(blur_sharpen_img))

%iii)Using the gaussianUnblur function - input to this is the blurred image
%created in part i
[output,mse_residuals,mse_original] = gaussianUnblur(blur_image,2,100,.0001);
%scale mse_original
current_max = max(mse_original);
current_min = min(mse_original);
mse_original = ((mse_original-current_min)*(255))/(current_max-current_min);

figure
plot(mse_residuals)
title('MSE - residuals')
ylabel('MSE')

hold on
plot(mse_original)
title('MSE - original')
ylabel('MSE')

im_noisy = single(imnoise(uint8(blur_image), 'gaussian'));
[output_noisy,mse_residuals,mse_original] = gaussianUnblur(im_noisy,2,100,.0001);
current_max = max(mse_original);
current_min = min(mse_original);
mse_original = ((mse_original-current_min)*(255))/(current_max-current_min);

figure
plot(mse_residuals)
```

```
title('MSE - residuals(noisy)')
ylabel('MSE')

hold on
plot(mse_original)
title('MSE - original(noisy)')
ylabel('MSE')

figure
subplot(2,3,1)
imshow(uint8(img))
title('original image')
subplot(2,3,2)
imshow(uint8(blur_image))
title('blurred image')
subplot(2,3,3)
imshow(uint8(output))
title('sharpened image')
subplot(2,3,4)
imshow(uint8(im_noisy))
title('blurred image(noisy)')
subplot(2,3,5)
imshow(uint8(output_noisy))
title('sharpened image(noisy)')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [I_k,MSE_residuals,MSE_orig] = gaussianUnblur(im_in,sigma,max_iter,t)

%I_0 is blurry input image
%I_k is the corrected image at iteration k
%G_sigma as defined in part i)

I_0 = im_in;

%Read in original image, only for purposes of calculating MSE
img = imread('brain.tif');
%img = single(img);

%Set initial I_k
I_k = im_in;

%Set padding
kernel = 6*sigma;
pad = floor((kernel-1)/2);
```

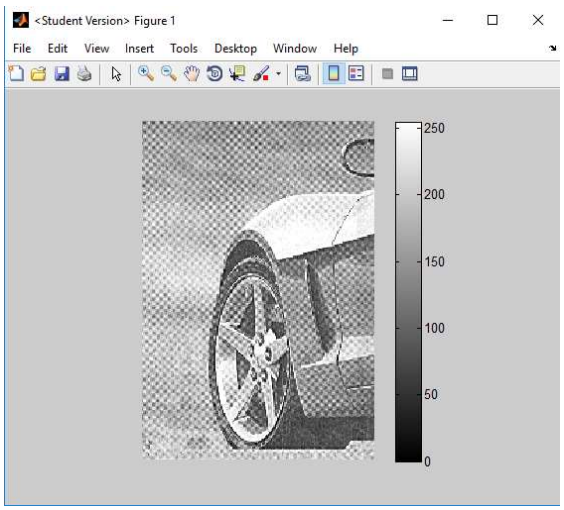
```
%Set filter
h = fspecial('gaussian',kernel,sigma);

%Allocate MSE arrays
MSE_residuals = [];
MSE_orig = [];

iterations = 0;
MSE_r = 1000;   %Just a dummy value
while MSE_r > t && iterations < max_iter
    I_k_prev = I_k;

    %(a) Compute A_k = I_k * G_sigma
    %[rows,cols] = size(I_k);
    %I_k = padarray(I_k,[pad pad],'symmetric');
    A_k = imfilter(I_k,h);
    %A_k = imcrop(A_k,[pad pad cols-1 rows-1]);
    %I_k = imcrop(I_k,[pad pad cols-1 rows-1]);

    %(b) Set B_k = I_0/A_k
    B_k = I_0./A_k;

    %(c) Compute C_k = B_k * G_sigma
    %B_k = padarray(B_k,[pad pad],'symmetric');
    C_k = imfilter(B_k,h);
    %C_k = imcrop(C_k,[pad pad cols-1 rows-1]);

    %(d) Set I_(k+1) = I_k x C_k
    I_k = I_k .* C_k;


    %Calculate MSE
    MSE_r = sum((sum(((I_k - I_k_prev ).^2))))/numel(I_k)
    MSE_residuals = [MSE_residuals, MSE_r];

    MSE_o = sum((sum(((img - I_k ).^2))))/numel(img)
    MSE_orig = [MSE_orig, MSE_o];

    %Increment iterations
    iterations = iterations+1;
end
```

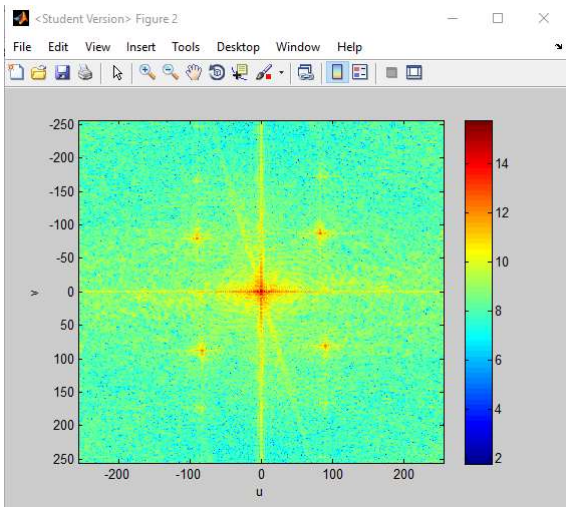
```
    function blurred_image = gaussianBlur(im_in,sigma)

[rows,cols] = size(im_in);

kernel = 6*sigma;
%Pad input image based on kernel size, so conextual region for edge pixels
%remains valid. Kernel size in always MxM

%Kernel will always be contered on pixel of interest. This means that the
%padding on each size has to be exactly (M-1)/2


pad = floor((kernel-1)/2);
padded_image = padarray(im_in,[pad pad],'symmetric');

%[rows,cols] = size(padded_image)

%Now lets actually create the Guassian Filter
%Of Form: h = fspecial('gaussian',hsize,sigma)
h = fspecial('gaussian',kernel,sigma);
blurred_image = imfilter(padded_image,h);

%imfilter automatically zero-pads the image, so its needs to be cropped
%to return to original size
blurred_image = imcrop(blurred_image,[pad pad cols-1 rows-1]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function blur_sharp_image = blurOrSharpen(im_in,sigma,w)

%We are implementing the following function
%I' = (1 + w) x I - w x I * G_sigma
%I * G_sigma convolution has already been done in the gaussianBlur()
%function , so we can use that

conv = gaussianBlur(im_in,sigma);

%Now we have the following operation:
%I' = (1 + w) x I - w x conv
blur_sharp_image = (1+w)*im_in-(w*conv);
```

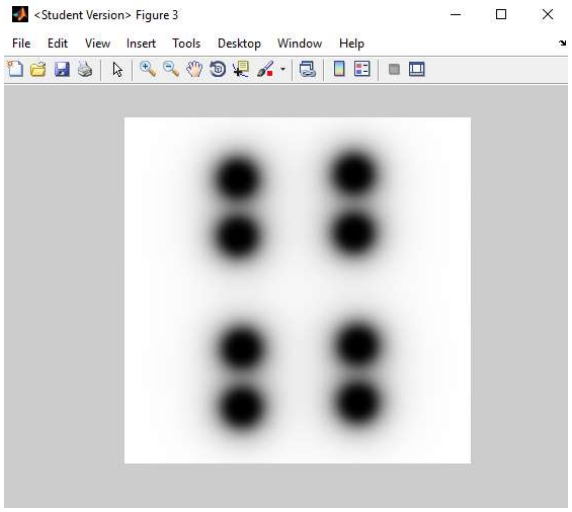
Problem 2: Butterworth Notch Reject Filtering in Frequency Domain

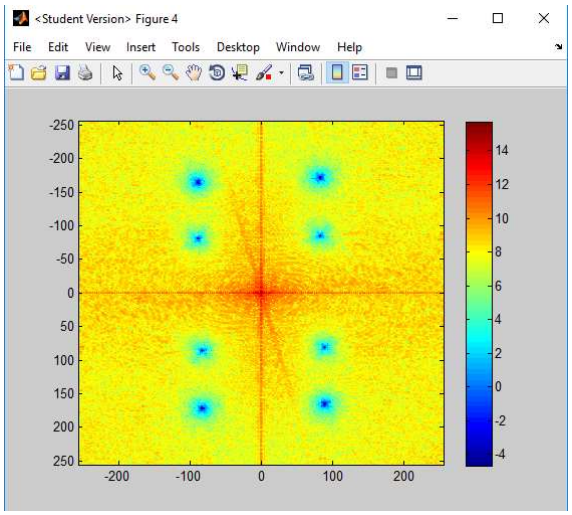
To run:
prob2main.m

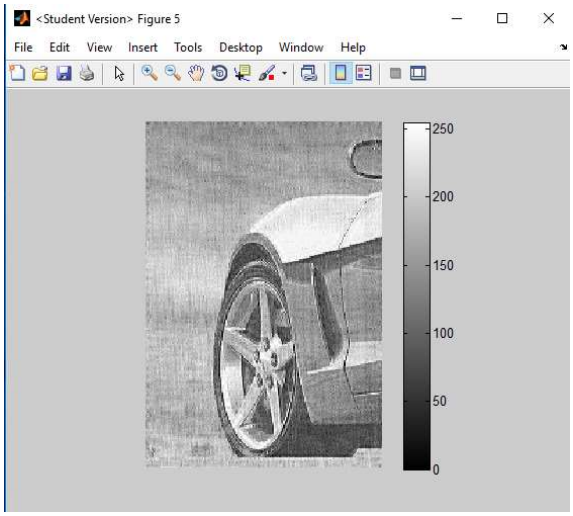
For car.tif:









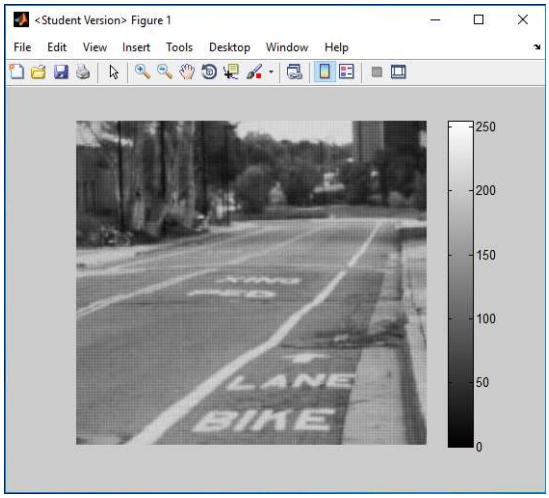


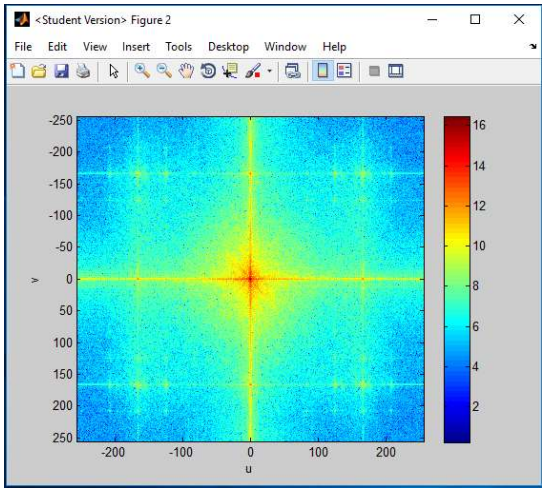
Parameters:

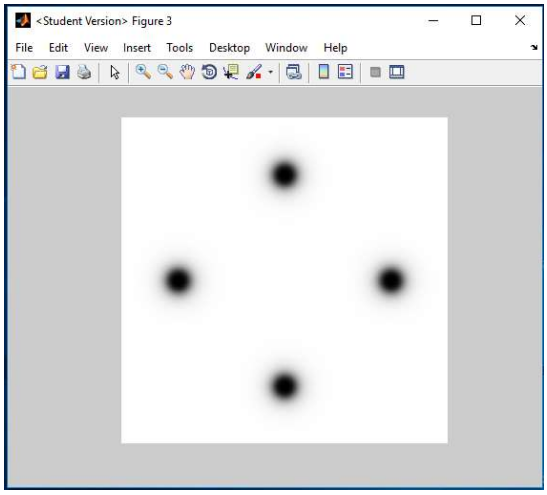
D_0 = 35, n = 2

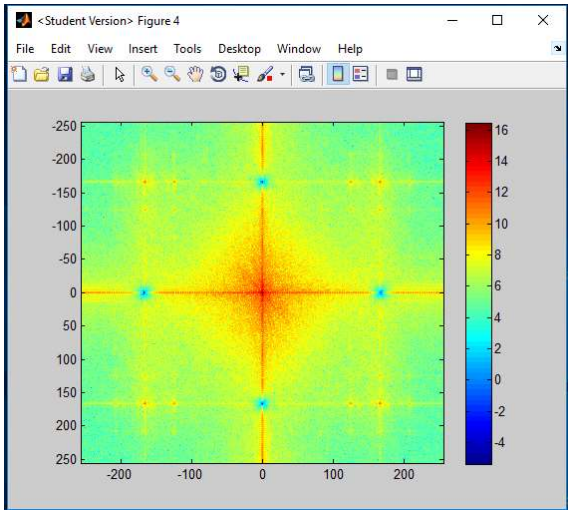
[u,v] → (-89,-165),(-89,-81),(-83,86),(-83,172)

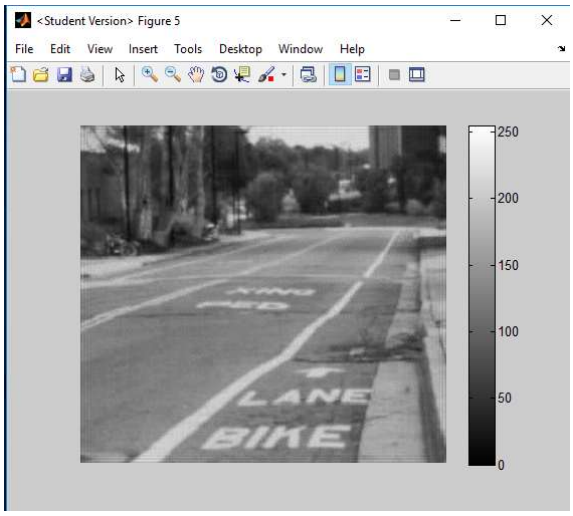
For street.png:











Parameters:

D_0 = 20, n = 2

[u,v] → (-166,0),(0,166)

```
%Problem 2: Butterworth Notch Reject Filtering in Frequency Domain

clear;
clc;


%Part
(i)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
img = imread('car.tif');
imshow(img)
colorbar;
r = 266;
c = 344;
pad_img = padarray(img, [r c],'post');

%Show FFT of image with "impulse-like" bursts
pad_img = fft2(pad_img);
pad_img = fftshift(pad_img);

figure
imagesc(-256:255,-256:255,log(abs(pad_img)));
colorbar;
xlabel('u');
ylabel('v');

[u,v]=meshgrid(-256:255);

%Implement the Notch reject filter
D_0 = 35;
n = 2;


u_k = -89;
v_k = -165;
D_K = ((u - u_k).^2 + (v - v_k).^2).^.5;
D_negK = ((u + u_k).^2 + (v + v_k).^2).^.5;
filt_p = 1./(1+(D_0./D_K).^(2*n));
filt_n = 1./(1+(D_0./D_negK).^(2*n));
filter1 = filt_p.*filt_n;

u_k = -89;
v_k = -81;
```

```
D_K = ((u - u_k).^2 + (v - v_k).^2).^.5;
D_negK = ((u + u_k).^2 + (v + v_k).^2).^.5;
filt_p = 1./(1+(D_0./D_K).^(2*n));
filt_n = 1./(1+(D_0./D_negK).^(2*n));
filter2 = filt_p.*filt_n;

u_k = -83;
v_k = 86;
D_K = ((u - u_k).^2 + (v - v_k).^2).^.5;
D_negK = ((u + u_k).^2 + (v + v_k).^2).^.5;
filt_p = 1./(1+(D_0./D_K).^(2*n));
filt_n = 1./(1+(D_0./D_negK).^(2*n));
filter3 = filt_p.*filt_n;

u_k = -83;
v_k = 172;
D_K = ((u - u_k).^2 + (v - v_k).^2).^.5;
D_negK = ((u + u_k).^2 + (v + v_k).^2).^.5;
filt_p = 1./(1+(D_0./D_K).^(2*n));
filt_n = 1./(1+(D_0./D_negK).^(2*n));
filter4 = filt_p.*filt_n;

figure
imshow(filter1.*filter2.*filter3.*filter4);

%Apply notch filter(s) to image
G = pad_img.*filter1.*filter2.*filter3.*filter4;
figure
imagesc(-256:255,-256:255,log(abs(G)));
colorbar;

reconstructed_img = ifft2(ifftshift(G));
reconstructed_img = reconstructed_img(1:end-r, 1:end-c);
reconstructed_img = uint8(reconstructed_img);
figure
imshow(reconstructed_img)
colorbar;

%Part
(2)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
img = imread('Street.png');
figure
```

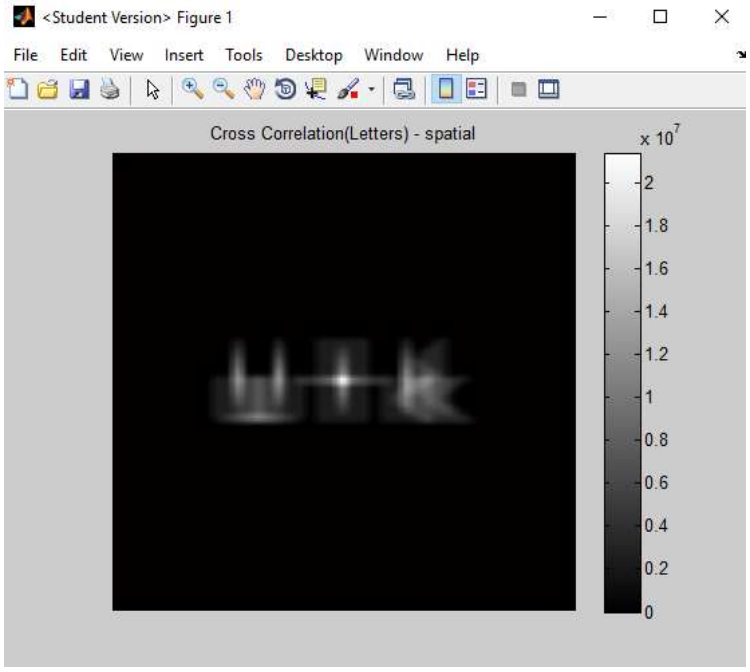
```
imshow(img)
colorbar;
r = 180;
c = 153;
pad_img = padarray(img, [r c],'post');

%Show FFT of image with "impulse-like" bursts
pad_img = fft2(pad_img);
pad_img = fftshift(pad_img);

figure
imagesc(-256:255,-256:255,log(abs(pad_img)));
colorbar;
xlabel('u');
ylabel('v');

[u,v]=meshgrid(-256:255);

%Implement the Notch reject filter
D_0 = 20;
n = 2;



u_k = -166.7;
v_k = 0;
D_K = ((u - u_k).^2 + (v - v_k).^2).^.5;
D_negK = ((u + u_k).^2 + (v + v_k).^2).^.5;
filt_p = 1./(1+(D_0./D_K).^(2*n));
filt_n = 1./(1+(D_0./D_negK).^(2*n));
filter1 = filt_p.*filt_n;

u_k = 0;
v_k = 166;
D_K = ((u - u_k).^2 + (v - v_k).^2).^.5;
D_negK = ((u + u_k).^2 + (v + v_k).^2).^.5;
filt_p = 1./(1+(D_0./D_K).^(2*n));
filt_n = 1./(1+(D_0./D_negK).^(2*n));
filter2 = filt_p.*filt_n;

figure
imshow(filter1.*filter2);

%Apply notch filter(s) to image
```

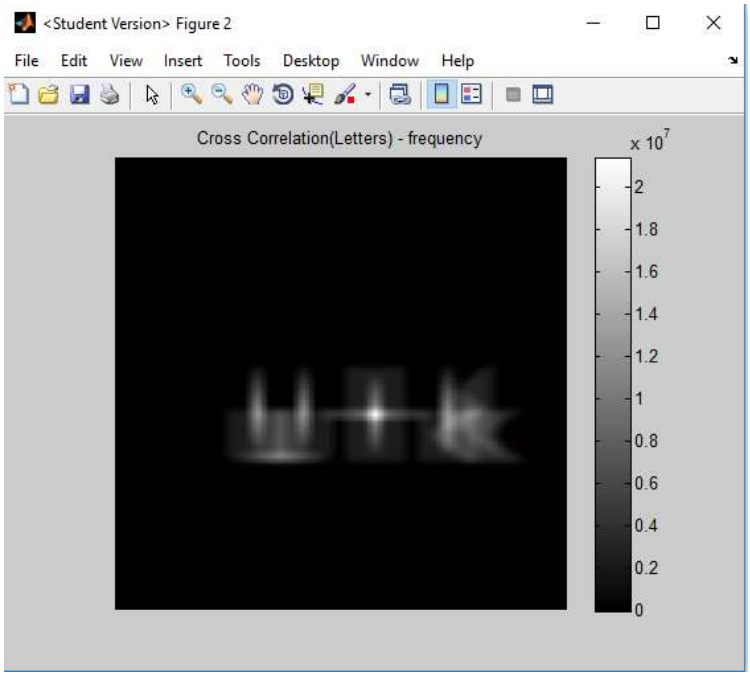
```
G = pad_img.*filter1.*filter2;
figure
imagesc(-256:255,-256:255,log(abs(G)));
colorbar;

reconstructed_img = ifft2(ifftshift(G));
reconstructed_img = reconstructed_img(1:end-r, 1:end-c);
reconstructed_img = uint8(reconstructed_img);
figure
imshow(reconstructed_img)
colorbar;
```

Problem 3: Template Matching

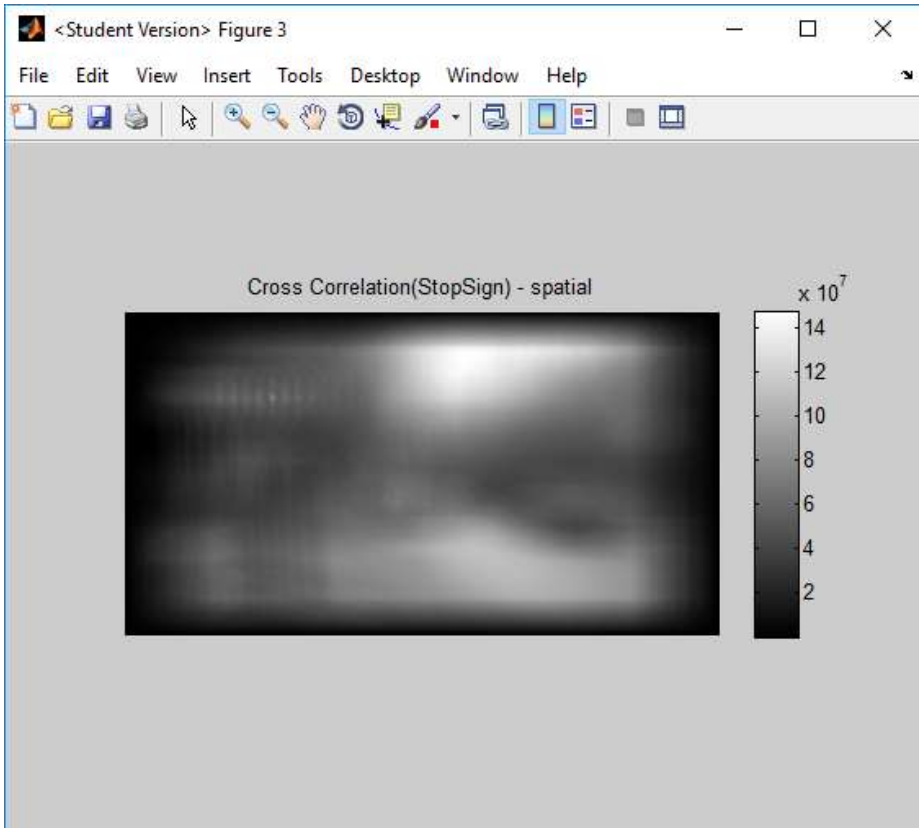
To run:
Prob3main.m

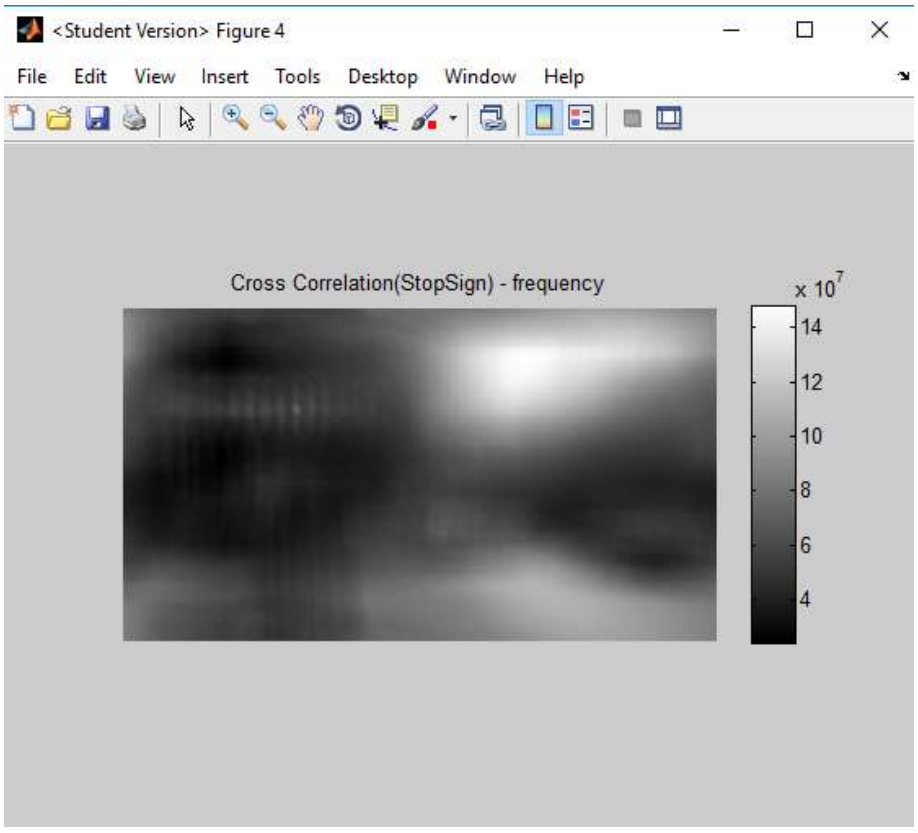
i) Cross-correlation filter on a simple image



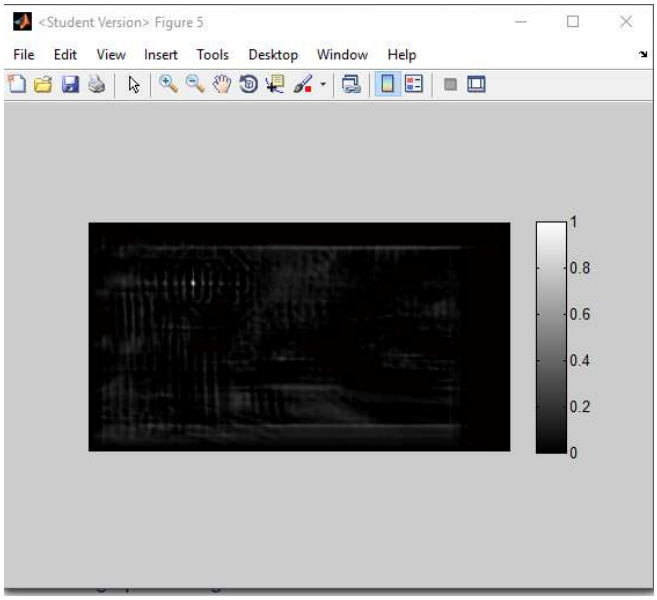


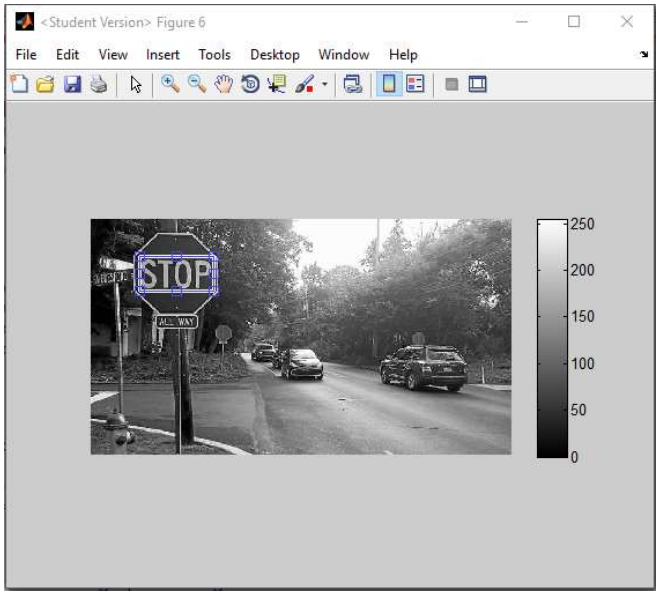
ii) Cross-correlation filter on a realistic image





iii) Normalized cross-correlation





iv) Regular cross-correlation does not take into account intensity patterns and/or illumination in the image. Given the template we have, if we tried to perform template matching with either a underexposed or overexposed image, it would likely fail. By normalizing the image before performing template matching, we take the illumination factor out of the result. This can be useful when performing template matching on a set of images taken during different parts of the day for example.

v) Template matching would likely fail in both these cases. The template in this case is simply a portion of the original image(or at least looks very similar to the sign in the image). When performing template matching, these pixel values of the template are compared to the corresponding values of pixels in the image. In the case that the the stop sign is a much

bigger/smaller part of the image, the pixel values simply won’t correspond to anything meaningful. Same in the case of if the stop sign is angled significantly. To solve this issue, one can add several variations of transformations of the template to the template matching algorithm, but a set of these must be available.

```
%Problem 3: Template Matching

clear;
clc;

%part i
img = imread('Letters.jpg');
img = double(img);
template = imread('LettersTemplate.jpg');
template = double(template);

%template_flip = flipdim(template,2);
template_flip = flipud(template);
template_flip = fliplr(template_flip);
C = conv2(template_flip, img);
imshow(C,[])
colorbar;
title('Cross Correlation(Letters) - spatial')


img_fft = fft2(img);
template_fft = padarray(template_flip, size(img_fft)-size(template),'post');
template_fft = fft2(template_fft);
test = img_fft .* (template_fft);
test = ifft2(test);
figure
imshow(test,[])
colorbar;
title('Cross Correlation(Letters) - frequency')

%part ii
img = imread('StopSign.jpg');
img = rgb2gray(img);
img = double(img);
template = imread('StopSignTemplate.jpg');
template = rgb2gray(template);
template = double(template);

template_flip = flipud(template);
template_flip = fliplr(template_flip);
C = conv2(template_flip, img);
figure
imshow(C,[])
```

```
colorbar;
title('Cross Correlation(StopSign) - spatial')


img_fft = fft2(img);
template_fft = padarray(template_flip, size(img_fft)-size(template),'post');
template_fft = fft2(template_fft);
test = img_fft .* (template_fft);
test = ifft2(test);
figure
imshow(test,[])
colorbar;
title('Cross Correlation(StopSign) - frequency')


%iii) Normalized Cross Correlation
ncc = normxcorr2(template,img);
figure
imshow(ncc)
colorbar;
[ypeak, xpeak] = find(ncc==max(ncc(:)));
yoffSet = ypeak-size(template,1);
xoffSet = xpeak-size(template,2);

figure
imshow(img,[]);
colorbar;
imrect(gca, [xoffSet+1, yoffSet+1, size(template,2), size(template,1)]);
```