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.

Zhouhang SHAO
A99086018

Problem 1:

(i) *gussianBlur* function:

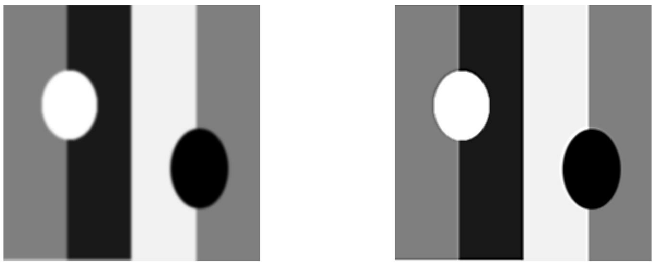
```
function  [result] = gussianBlur(im_in, sigma)
    h = fspecial('gaussian', 6*sigma, sigma);
    result = imfilter(im_in, h, 'conv', 'circular');
end
```

(ii) *blurOrSharpen* function:

```
function [output] = blueOrSharpen(in_im, w, sigma)
    if(w < -1 || w > 1)
        disp('Invalid w value');
        output = in_im;
    end
    input = in_im;
    output = (1 + w).*input - w.* gaussianBlur(input,sigma);
end
```

(a) What values of weight w give the most blurring and most sharpening? Explain why for both cases.

- When w equals to -1, the algoirhtm generates the most blur result image
- When w equals to 1, the algorithm generates teh most sharpening image
- The result images is composited from two parts, the orignal image and an unshapen mask w(I - I * G). When w is positive, the result image is made of the original image plus unsharpen mask(edges) to the original image. Thus, result image becomes sharpen compared to the original one.
- when w is negative, the result image is generated by subtracting the unshapen mask(edges) from the original image, thus generates a blurred version of the original image
- In the following three images, the top one is the original image in uint8 form.
- On the bottom line, the left image corresponds to the most blurred image when w = -1 and the right image represents the image for w = 1



(b) Run your filter on the image with the maximal amount of blurring (i.e., the w value determined above) and with sigma = 10. Now try to reverse this by sharpening the image with the maximal amount of sharpening (the w from above) and the same sigma = 10. Did you recover the original image? Why or why not?

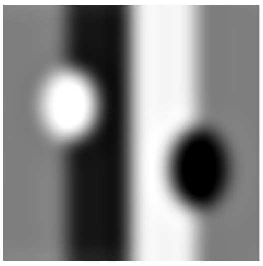
I cannot recover the original image.

After blur:

$$I' = I * G_\sigma$$

Sharpen the blur image

$$I'' = 2I' - I' * G_\sigma I'' = 2I * G_\sigma - I * G_\sigma * G_\sigma \neq I$$

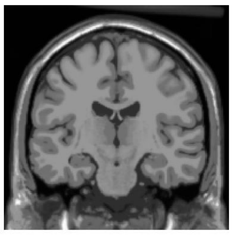


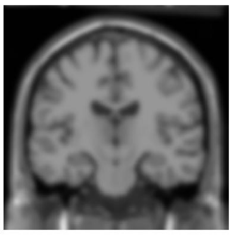
(c) Run your filter on the image with w = 1 and sigma = 10. Describe the artifacts that you see in the filtered image. Why does this happen?

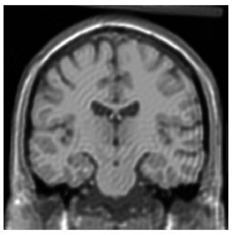
I see some white halo around black cicles and some black halo around white circles. I think the possible reason is that the signma value is too large. Thus, the unsharp mask will have some intensity peeks on the edges. When I change sigma value to 1, the result image looks much better

(iii)

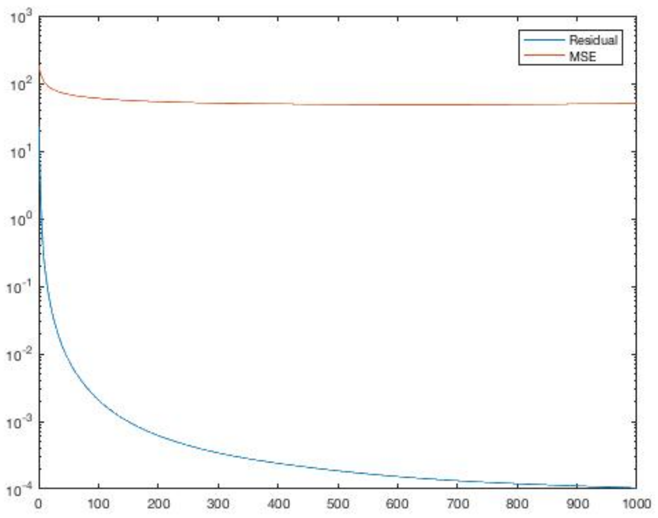
(a) GaussianUnblur function:





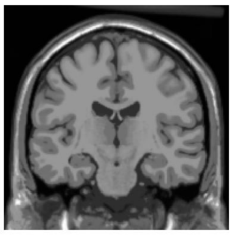


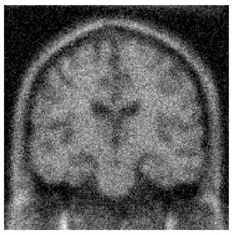
Using the following gaussianUnblur method, we can enhance a blurred image(bottom left) and getting a sharpen verson(bottom right). The original image is display on the top. As we can see, the recovered image definitly looks more detailed and clear compared to the blurred input image. However, compared to the original image, we can still observe noise and artifacts. If we look at the following graph, we will find although the residual gets smaller at each iteration the MSE is still relatively large compared to the original image.

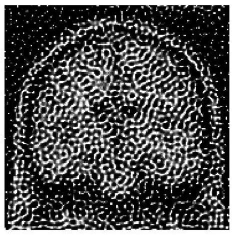


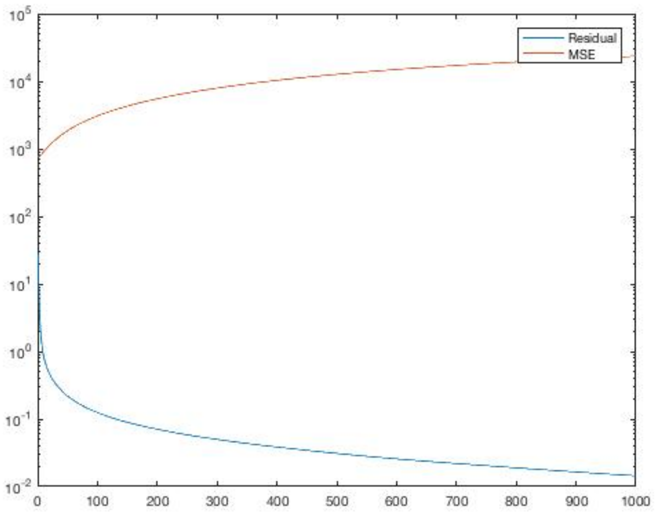
```matlab
function [output] = gaussianUnblur(im_in, sigma, max_iter, t)
    k = 0;
    im_prev = im_in;
    [r,c] = size(im_in);
    while(k < max_iter)
        A = gaussianBlur(im_prev,sigma);
        B = im_in./A;
        C = gaussianBlur(B,sigma);
        im_curr = im_prev .* C;
        k = k + 1;
        MSE = sum(sum((im_curr - im_prev).^2))/(r*c);
        if (MSE < t)
            disp('converge')
            break;
        end
        im_prev = im_curr;
    end
    output = im_curr;
end

function gaussianUnblurHelper(orig, im_in, sigma, max_iter, t)
    k = 1;
    im_prev = im_in;
    [r,c] = size(im_in);
    while(k <= max_iter)
        A = gaussianBlur(im_prev,sigma);
        B = im_in./A;
        C = gaussianBlur(B,sigma);
        im_curr = im_prev .* C;

        MSE = sum(sum((im_curr - im_prev).^2))/(r*c);
        res_l(k) = MSE;
        mse_l(k) = sum(sum((im_curr - orig).^2))/(r*c);
        k = k + 1;
        if (MSE < t)
            disp('converge')
            break;
        end
        im_prev = im_curr;
    end

    %plot
    figure,semilogy(1:k-1, res_l)
    hold on;
    semilogy(1:k-1, mse_l);
    legend ('Residual','MSE') ;
    hold off;
end
```

(b) recover the noise blur image









Repeating the steps in the previous question, but using a nosiy image, we get the results as shown in the above resulting images.

As we can see, our recover algorithm does not handle the noise properly. Instead of improving the image quality, the unblur algorithm decreases the image quality and amplify the noises in the input image. A lot of details lost during this process. If we look at the plotted graph, the residual seems to converge but for MSE, instead of decreasing, it increases at each iteration. The increasing MSE also suggests a very bad image quality for the result image.

Source code:

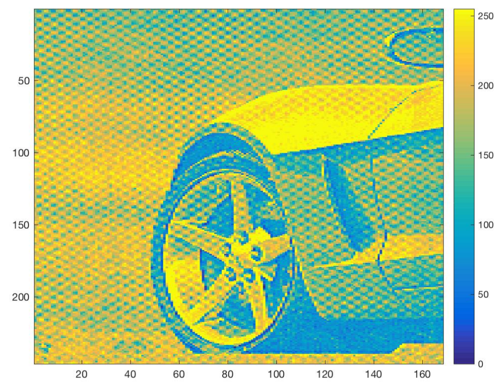
```
image = imread('pattern.tif');
imwrite(uint8(image),'origin.jpg');
%Problem 1-ii-b
mostSharpen = blurOrSharpen(image, 1,1);
mostBlurred = blurOrSharpen(image,-1,1);

tosave = figure;
(imshow(uint8(mostBlurred)));
```

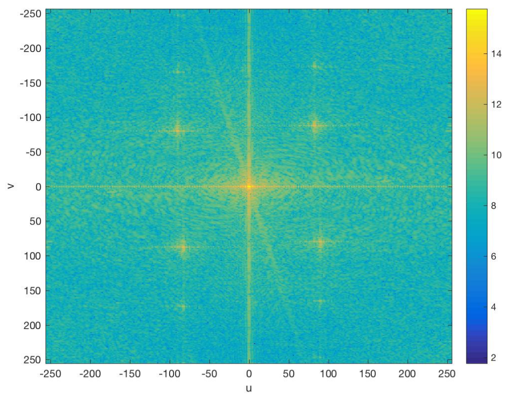
```
saveas(tosave,'mostBluredSigma10.jpg')

tosave = figure;
imshow(uint8(mostSharpen));
saveas(tosave,'mostSharpenSigma10.jpg')

tosave = figure;
blurred = blurOrSharpen(image,-1,10);
imshow(uint8(blurOrSharpen(blurred,1,10)));
saveas(tosave,'sharpenOnBlurImageSigma10.jpg');

%Problem 1-ii-c
result = blurOrSharpen(image,1,10);
tosave = figure;
imshow(uint8(result));
saveas(tosave,'sharpenSigma10.jpg');

% problem 1-iii-a

brain = imread('brain.tif');
tosave = figure;
imshow(uint8(brain));
saveas(tosave,'brainOrigin.jpg')
toRecover = gaussianBlur(brain,2);
tosave = figure;
imshow(uint8(toRecover));
saveas(tosave,'brainBlurred.jpg');
recover = gaussianUnblur(toRecover,2,1000,0.0001);
tosave = figure;
imshow(uint8(recover));
saveas(tosave,'brainSharpen.jpg');
gaussianUnblurHelper(brain,toRecover,2,1000,0.0001);

%problem 1-iii-b
toRecover = single(imnoise(uint8(toRecover), 'gaussian'));
tosave = figure;
imshow(uint8(toRecover));
saveas(tosave,'brainBlurredNoise.jpg');
recover = gaussianUnblur(toRecover,2,1000,0.0001);
tosave = figure;
imshow(uint8(recover));
saveas(tosave,'brainSharpenNoise.jpg');
gaussianUnblurHelper(brain,toRecover,2,1000,0.0001);
```

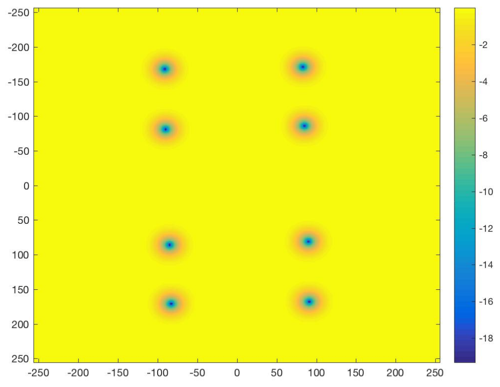
Problem 2:

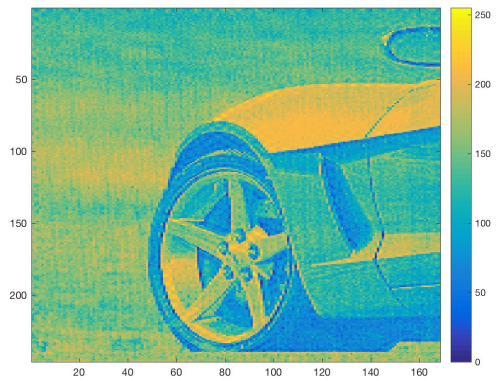
(i)

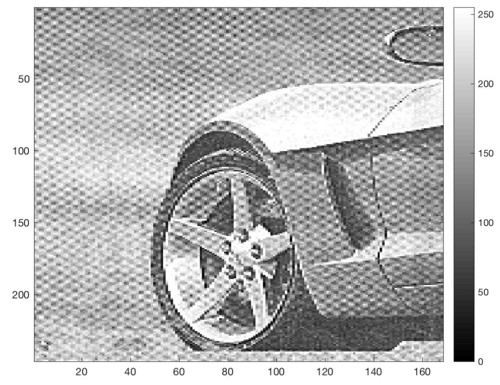
- n = 3, D0 = 25
- (-90,-81), (-91,-168), (-85,86), (-83,171)

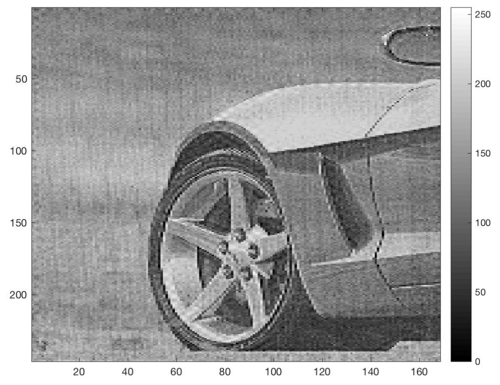










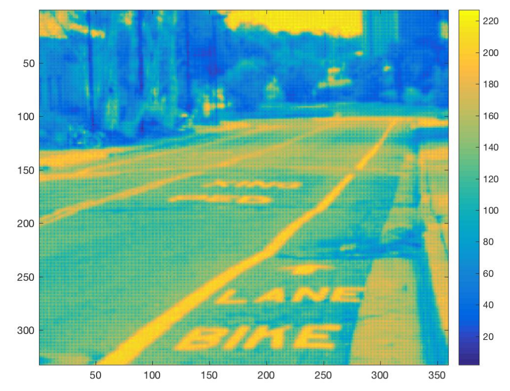


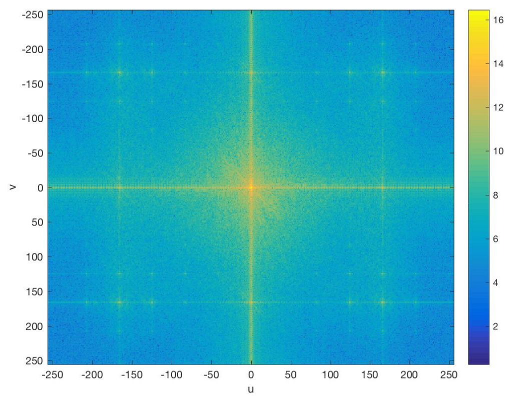
Source Code:

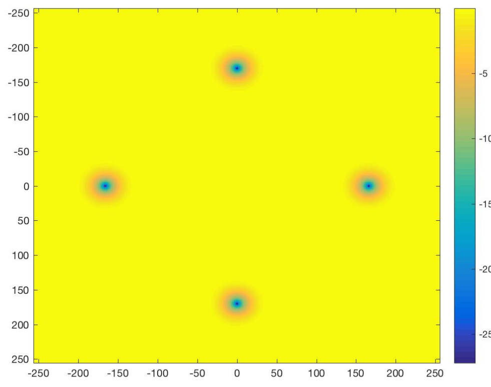
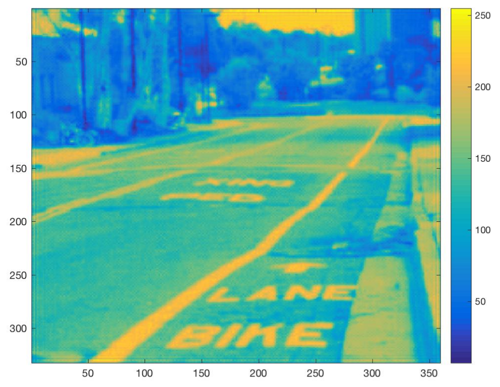
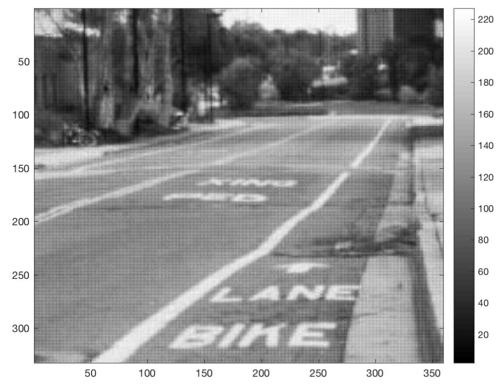
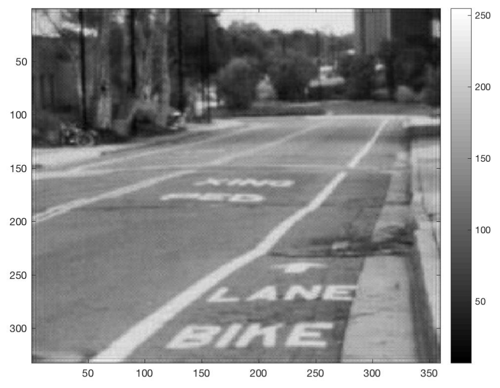
```matlab
function [imFFT, filteredImage, notchFilter,filterSpatialImage] = BNRFilter(n,r,uk,vk,input)
    fftCar = fft2(input,512,512);
    imFFT = fftshift(fftCar);

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

    notchFilter = ones(size(u,1),size(u,2));
    for i = 1 : size(uk,2)
        dkPos = sqrt((u-uk(i)).^2 + (v-vk(i)).^2);
        dkNeg = sqrt((u+uk(i)).^2 + (v+vk(i)).^2);
        notchFilter = notchFilter .* (1./(1+((r./dkPos).^(2*n)))) .* (1./(1+((r./dkNeg)).^(2*n)));
    end

    % filter the image in frequency domain
    filteredImage = fftCar .* fftshift(notchFilter);
    filterSpatialImage = ifft2(filteredImage);
    maxIntensity = max(filterSpatialImage( : ));
    filterSpatialImage = uint8((filterSpatialImage./maxIntensity)*255);
    filterSpatialImage = filterSpatialImage(1 : size(input,1),1:size(input,2));
end

car = imread('car.tif');
tosave = figure;
imagesc(uint8(car));colorbar;
saveas(tosave, 'origin_car.jpg')
uk = [-90,-91,-85,-83];
vk = [-81,-168,86,171];
[imFFT, filteredImage, notchFilter, filterSpatialImage] = BNRFilter(3,25,uk,vk,car);
tosave = figure;
imagesc(-256:255,-256:255,log(abs(imFFT))); colorbar;
xlabel('u'); ylabel('v');
saveas(tosave,'imFFT_car.jpg');
tosave = figure;
imagesc(filterSpatialImage);colorbar
saveas(tosave,'filterSpatialImage_car.jpg');
tosave = figure;
imagesc(filterSpatialImage);colormap gray;
saveas(tosave,'filterSpatialImage_car_gray.jpg');
tosave = figure;
imagesc(-256:255,-256:255,log(abs(notchFilter))); colorbar;
saveas(tosave,'notchFilter_car.jpg');
tosave = figure;
imagesc(-256:255,-256:255,log(abs(fftshift(filteredImage)))); colorbar;
saveas(tosave,'filteredImage_car.jpg');
```

(ii)

- n = 4, D0 = 30
- (0,-170), (166,0)





Source code:

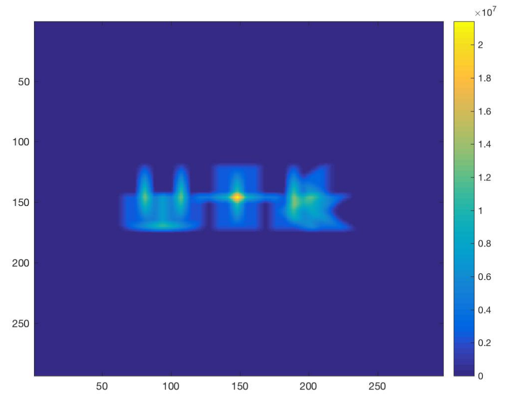
```
function [imFFT, filteredImage, notchFilter,filterSpatialImage] = BNRFilter(n,r,uk,vk,input)
    fftCar = fft2(input,512,512);
    imFFT = fftshift(fftCar);

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

    notchFilter = ones(size(u,1),size(u,2));
    for i = 1 : size(uk,2)
        dkPos = sqrt((u-uk(i)).^2 + (v-vk(i)).^2);
        dkNeg = sqrt((u+uk(i)).^2 + (v+vk(i)).^2);
        notchFilter = notchFilter .* (1./(1+((r./dkPos).^(2*n)))) .* (1./(1+((r./dkNeg)).^(2*n)));
    end

    % filter the image in frequency domain
    filteredImage = fftCar .* fftshift(notchFilter);
    filterSpatialImage = ifft2(filteredImage);
    maxIntensity = max(filterSpatialImage( : ));
    filterSpatialImage = uint8((filterSpatialImage./maxIntensity)*255);
    filterSpatialImage = filterSpatialImage(1 : size(input,1),1:size(input,2));
end

street = imread('street.png');
tosave = figure;
imagesc(street);colorbar;
saveas(tosave, 'origin_street.jpg')
uk = [0,166];
vk = [-170,0];
```

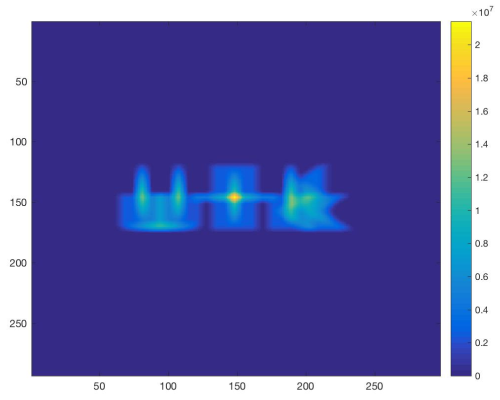
```matlab
[imFFT, filteredImage, notchFilter, filterSpatialImage] = BNRFilter(4,30,uk,vk,street);

tosave = figure;
imagesc(-256:255,-256:255,log(abs(imFFT))); colorbar;
xlabel('u'); ylabel('v');
saveas(tosave,'imFFT_street.jpg');
tosave = figure;
imagesc(filterSpatialImage);colorbar
saveas(tosave,'filterSpatialImage_street.jpg');
tosave = figure;
imagesc(filterSpatialImage);colormap gray;
saveas(tosave,'filterSpatialImage_street_gray.jpg');
tosave = figure;
imagesc(-256:255,-256:255,log(abs(notchFilter))); colorbar;
saveas(tosave,'notchFilter_street.jpg');
tosave = figure;
imagesc(-256:255,-256:255,log(abs(fftshift(filteredImage)))); colorbar;
saveas(tosave,'filteredImage_street.jpg');
```

Problem 3:

(i):

- Convolution in spatial domain, bottom left
- Multiply in frequency domain, bottom right



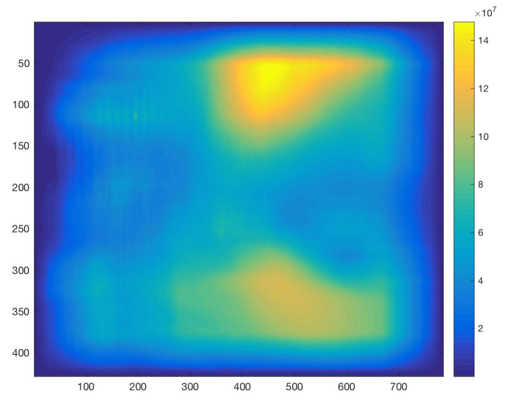


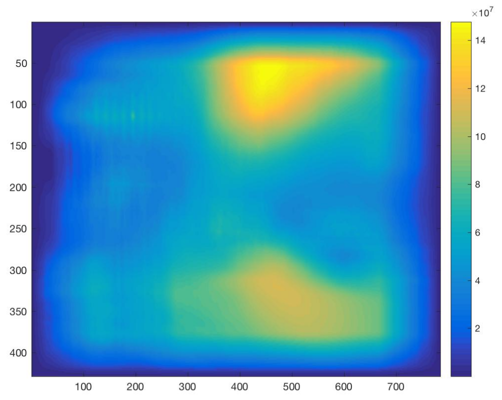
source code:

```matlab
% problem 3 i
letter = double(imread('Letters.jpg'));
template = double(imread('LettersTemplate.jpg'));
convResult = convolveSpatial(letter,template);
tosave = figure;
imagesc(convResult),colormap gray;
saveas(tosave, 'convolution_spatial_letter.jpg');
filteredImage = multiFreq(letter, template);
tosave = figure;
imagesc(filteredImage),colormap gray;
saveas(tosave, 'convolution_freq_letter.jpg');

function [convResult] = convolveSpatial(image, template)
    % rotate before convolution
    rotTemplate = imrotate(template,180);
    % convolution in spatial domain
    convResult = conv2(image, rotTemplate);
end

function [filteredImage] = multiFreq(image, template)
     rotTemplate = imrotate(template,180);
    [rl, cl] = size(image);
    [rt, ct] = size(template);
    fftImage = fft2(image, rl+ rt - 1, cl + ct - 1);
    fftTemplate = fft2(rotTemplate, rl + rt - 1, cl + ct - 1);
    % multiply in the freq domain
    ffImage = fftImage .* fftTemplate;
    filteredImage = ifft2(ffImage);
end
```

(ii)

- Convolution in spatial domain, bottom left
- Multiply in frequency domain, bottom right

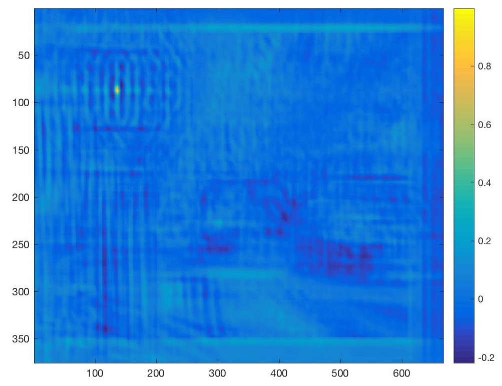


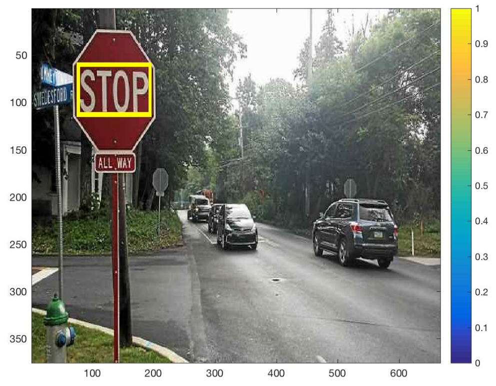


Source code:

```matlab
% problem 3 ii
rgbImage = imread('StopSign.jpg');
stopSign = double(rgb2gray(rgbImage));
stopSignTemp = double(rgb2gray(imread('StopSignTemplate.jpg')));
convResult = convolveSpatial(stopSign,stopSignTemp);
tosave = figure;
imagesc(convResult),colormap gray;
saveas(tosave, 'convolution_spatial_sign.jpg');
filteredImage = multiFreq(stopSign, stopSignTemp);
tosave = figure;
imagesc(filteredImage),colormap gray;
saveas(tosave, 'convolution_freq_sign.jpg');

function [convResult] = convolveSpatial(image, template)
    % rotate before convolution
    rotTemplate = imrotate(template,180);
    % convolution in spatial domain
    convResult = conv2(image, rotTemplate);
end

function [filteredImage] = multiFreq(image, template)
     rotTemplate = imrotate(template,180);
    [rl, cl] = size(image);
    [rt, ct] = size(template);
    fftImage = fft2(image, rl+ rt - 1, cl + ct - 1);
    fftTemplate = fft2(rotTemplate, rl + rt - 1, cl + ct - 1);
    % multiply in spatial domain
    ffImage = fftImage .* fftTemplate;
    filteredImage = ifft2(ffImage);
end
```

(iii)





Source code:

```matlab
%problem 3 iii
resultImage = normxcorr2(stopSignTemp, stopSign);
[h,w] = size(stopSign);
[r,c] = size(stopSignTemp);

%crop the image to original size
resultImage = resultImage(1 + floor(r/2): floor(r/2) + h, 1 + floor(c/2) : floor(c/2) + w);
tosave = figure;
imagesc(resultImage),colorbar;
saveas(tosave, 'normalized_cross_correlation.jpg');

maxInt = max(resultImage(:));
[x1,y1]=find(resultImage == maxInt);
stopSign = insertShape(rgbImage, 'rectangle', [round(y1-c/2) round(x1-r/2), c, r],
'LineWidth',5);
tosave = figure;
imagesc(stopSign);
saveas(tosave, 'detected_origin_image.jpg');
```

(iv)Why is normalized cross-correlation more effective compared to cross-correlation for template matching?

Correlation measures how similar the template and the target subimage we want to compare. It works well when the mignitude of two images are similar. However, the traditional cross-correlation does not work well when the two images have very different magnitudes in pixel intensity. Cross-correlation takes into count the mean image intensities and dividing by the std. Those, it will more robust for varies input image and give better matching results.

(v) Using the same template StopSignTemplate.jpg on another image with a stop sign taken at a much different camera angle or much closer/farther away, would normalized still perform as well? Why or why not?

We will not have a good template matching result when the tempelate and objects looks quite different.

cross-correlation based image is a naive template matching. It evalute the similarity of two images by comparing the template and target image pixel by pixel. For example, if the input image is rotated,, pixels mapping of the two images are no long valid. Thus, cross-correlation based template matching can genrate very bad result when the template and objects are quite different.