DEBLURRING IMAGES USING CEPSTRUM ANALYSIS

Submitted to

Dr. Becker

Prepared by

Fikadu Dagefu
Nathan Shepard

PHY 333 Semester Project
University of Texas at Austin
Department of Electrical and Computer Engineering
SPRING 2006

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

This report discusses a research project that investigated the use of Fourier optics to correct blurring of images caused by motion. More specifically, linear motion blur of images such as in photographing a speeding car or a shooting star has been considered. We have considered three different cases of linear motion blur and investigated ways of applying inverse filtering techniques to correct the blur. Following are the three cases that we investigated.

2.0 OBJECTIVES

2.1 Known Blur Angle and Length

First we considered the simplest case where we assumed that we have a prior knowledge of the length and angle by which the image has been blurred. Assuming o(x,y) is the original image, b(x,y) and h(x,y) are the blurred and the point spread function respectively, we can relate the two images as follows.

$$B(x, y) = O(x, y)H(x, y)$$

Where the functions with the capital letters represent the Fourier transforms. At first blush it seems like all we do is divide the blurred image by the Fourier transform of the point spread function H(x, y). This function obviously could have zeros at several spatial frequencies which suggests the solution to the problem is not merely dividing by H(x, y). This problem has been investigated and our solution has been discussed.

2.2 Unknown Blur Angle and Length

Secondly, we approached the problem with no prior knowledge of the blurring. For this case, the only information we have is the blurred image. In this case, since the direction and the blurring extent is not known we will have to somehow estimate these values from the degraded image and try to correct the blur using those values. One approach is to use the cepstrum (which is like the spectrum of a spectrum with some adjustments) to find the periodicity and thus help us determine the extent of blur [1]. We have used this approach to estimate amount of blur. We have tried to solve the problem in two ways.

The first method is the direct method which simply estimates the most likely blurring and tries to correct it once. The other method is correcting the blur iteratively which by estimating the blur and improving the image for each estimate using binary user feedback.

2.3 Unknown Blur Angle and Length, with the Addition of Gaussian Noise

Lastly, we investigated the case where there is noise in addition to linear motion blur. We investigated how the situations discussed above in the presence of noise and how we can improve our solutions for cases 2 and 3 when noise is added. Minimization of mean-squared error was one of the major objective criteria we considered when optimizing the restoration of the image.

3.0 APPROACH AND TECHNIQUES

3.1 Blurring and Test images

We wrote a Matlab code that degrades (blurs) the image. Linear motion blur is equivalent to convolving the image with a rect() function. In Matlab, we wrote a function that takes a length and an angle and outputs the blurred image. We made use of the "fspecial" function in Matlab to create the point spread function (PSF) which is the blurring filter.

3.2 Unknown Blur Length , Noise Free

When the angle or the length of the blur is unknown in a given image (See Figure 1), it is possible to estimate this angle quite accurately using analysis in the Cepstrum domain. To get to the Cepstrum domain, we started by finding the magnitude of the 2-D discrete Fourier transform of the original image. Next, we took the logarithm of the frequency spectrum and squared it, which highlighted the most powerful frequencies and diminished the power at DC (See Figure 2). Taking the 2-D inverse Fourier transform of this space yielded the image in the Cepstrum domain (See Figure 3), and we were able to use edge detection and the Hough transform to find the most likely blur angles (See Figure 4).

Whether we are given the blur angle or we have estimated it, for a given angle we can estimate the blur length in an image. With the image in the Cepstrum domain, we rotate the image by the expected blur angle and then take the average of each column. By finding the number of columns between zero crossings, we were able to find the periodicity and estimate the blur length for a given angle with a fairly high accuracy.

We wrote a Matlab program that estimates the 20 most likely blur angles and estimates the blur length for each. It then displays each image in the order of its blur angle likelihood. For images clearly blurred along one angle, this algorithm works very well, but it does require human input to determine whether each result is an improvement or not.

While testing this algorithm with known blur angles, we found that angle estimates within 3 degrees of the actual blur angle yield visually appreciable results. This in mind, we wrote an additional Matlab program that performs blur length estimation for each of 36 angles spaced evenly every 5° over the 180° possibilities. Afterwards, the program accepts user input as to which image is the most visually cohesive. The program then iterates over 20 angles spaced evenly every 2° centered around the last angle chosen. For resolution purposes, the program iterates on the user's choice once more, with 20 angles spaced evenly every 0.5° centered around the user's last chosen angle.

3.3 Noise in the Deblurring Algoorithm

High-frequency or impulse shaped ("salt and pepper") noise causes errors in image representation in the Cepstrum domain by adding power to higher frequencies that should not be in the blurred, inherently low-frequency original image. This can cause the Hough transform to miss important edges in the Cepstrum domain, and it is difficult to estimate either the blur angle or the blur length. Although applying a low-pass or averaging filter the original image is not advisable in a deblurring algorithm, a non-linear noise-canceling filter such as a median filter will eliminate noise without further blurring the image. We employed a median filter in our blur angle and blur length estimation routines with great success.

4.0 RESULTS

4.1 Known Blur Length, Noise Free

In this case since we already know the extent and orientation of blur we could just use those values to undo the blur. This is the equivalent of inverse filtering and in frequency domain it's equivalent to dividing by the Fourier transform of the point spread function which is called optical transfer function (OTF). The only problem that arises in taking this approach is the fact that the OTF could have zeros and we can't divide by zero. Here we should note that the image can not be recovered completely because some of the frequencies in the original image have been zeroed out even in the noise free case [2]. In matlab, we set up a 'for' loop that looks for the zero frequencies in the OTF and sets them to a very small number (in the order of 10^{-6}). This solves the divide by zero problem.

4.2 Unknown Blur Length, Noise Free

Since we are estimating the blur length and angle in the case where we don't have a prior knowledge of the extent of the blur, there needs to be a way to measure how good our estimates are. As described in Sondhi[2] , one way to approach this problem is to minimize the mean-squared error which in essence is minimizing the discrepancy between the original and the restored image. Another method we considered was to use user feedback to improve the deblurring process. We can do the blur estimation not just once but iteratively using user input (See Figures 1-4).

Our most successful algorithm is the iterative routine, which divides the polar space into 36 even angles and estimates the blur length and deblurred image for each (See Figure 1). Once the user chooses the preferred image, in our example 26°, the algorithm displays 20 more images 2° apart around that angle (See Figure 2). The user chooses 22° and the process repeats with 0.5° resolution (See Figure 3). At this resolution, very little

differences were observed between neighboring images, and higher resolution did not prove to be worthwhile. Therefore, the program simply displays the best image at this point, in our example, 23° blur angle and 21 pixel blur length (See Figure 4).

4.3 Effect of Noise in the Deblurring Algorithm

In the presence of noise, it's obvious that it's not easy to get the deblurring algorithm to improve the blurred image as good as the noise free case. After making sure that our algorithm was working for the noise free case, we introduced noise in addition to the motion blur. We used a non-linear noise canceling filter (median filter) to eliminate the noise. As expected, the blurred image (which also has noise) improves a lot after we applied our deblurring algorithm to it (See Figure 9).

5.0 CONCLUSION

In this report, we described our research project that explored the use Fourier optics to correct blurring of images caused by motion. The report describes how we went about deblurring the images in three cases. These were when we know the extent of blur and without noise, when we don't have prior knowledge of the blurring and without noise and the above cases in the presence of noise. The report also discussed the approach and techniques we used to solve the problem and results of the project. We have attached several test images and the source code.

6.0 REFERENCES

[1] Teaching Biological Oscillators sustainability -Breakthru in Measuring Internal Coherence.http://www.soulinvitation.com/cepstrum/index.html, accessed 4/21/2006.

[2] Man Mahan Sondhi , "Image restoration: The removal of spatially invariant degradation'' in Proceedings of the IEEE, Vol 6, No 7, July 1972

[3] Becker, Michael, Michael Becker's Home Page, http://www.ece.utexas.edu/~becker, accessed 4/26/2006.

APPENDIX A – IMAGES AND FIGURES

APPENDIX A – IMAGES AND FIGURES

Figure 1. Input Image with Known 10 Pixel, 45° Blur

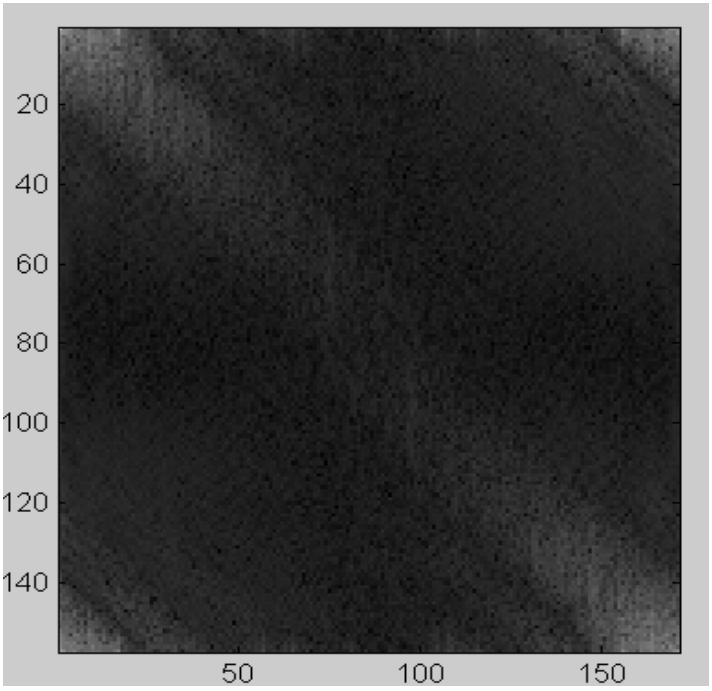

Figure 2. Log Magnitude Squared Frequency Domain of Known 10 Pixel, 45° Blur

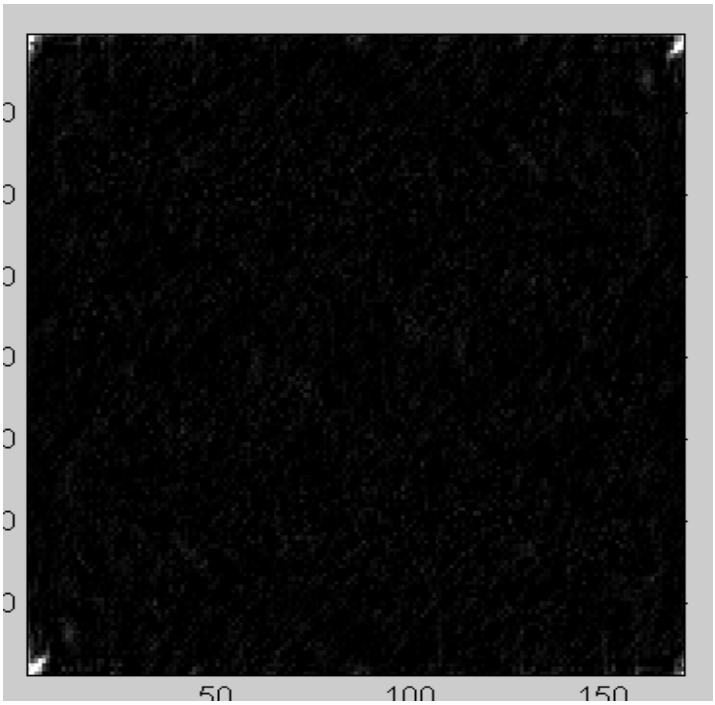

Figure 3. Cepstrum Domain with a Known 10 Pixel, 45° Blur

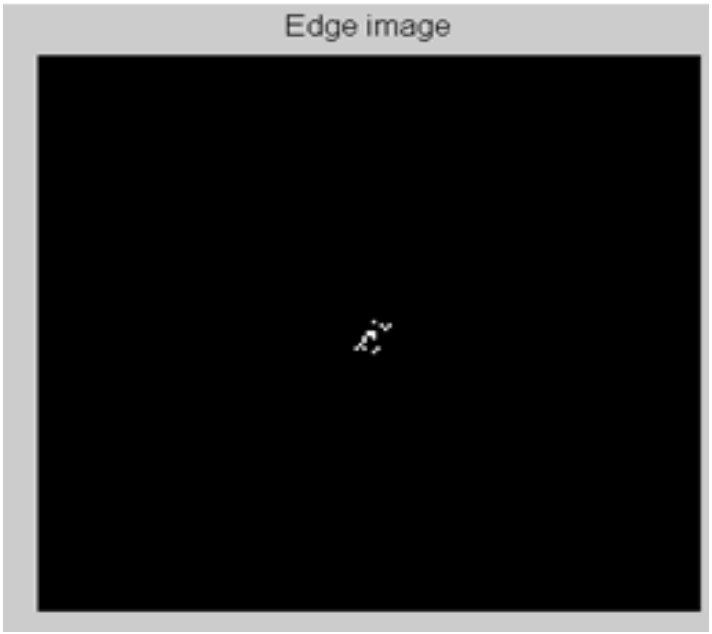

Figure 4. Resulting Edge Image is Followed by the Hough Transform

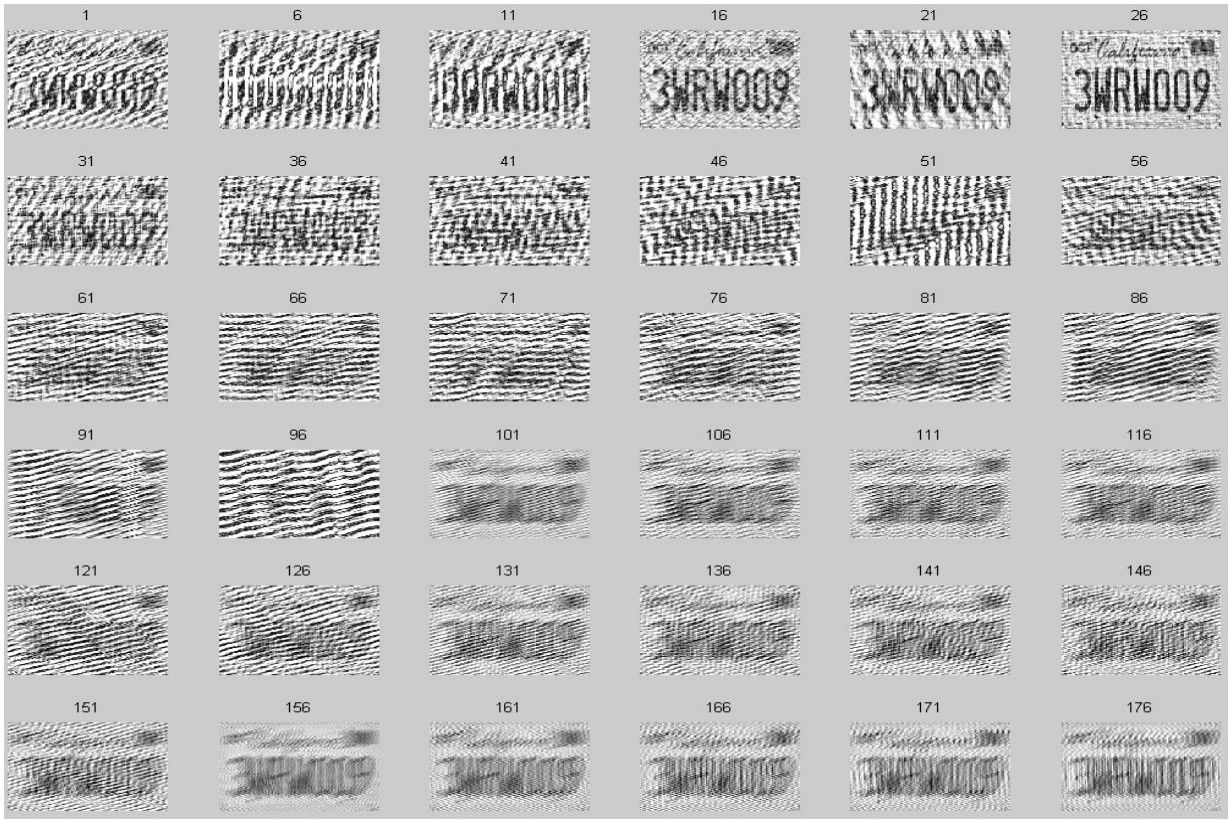

Figure 5. 1st Iteration of User-Assisted Deblurring Algorithm

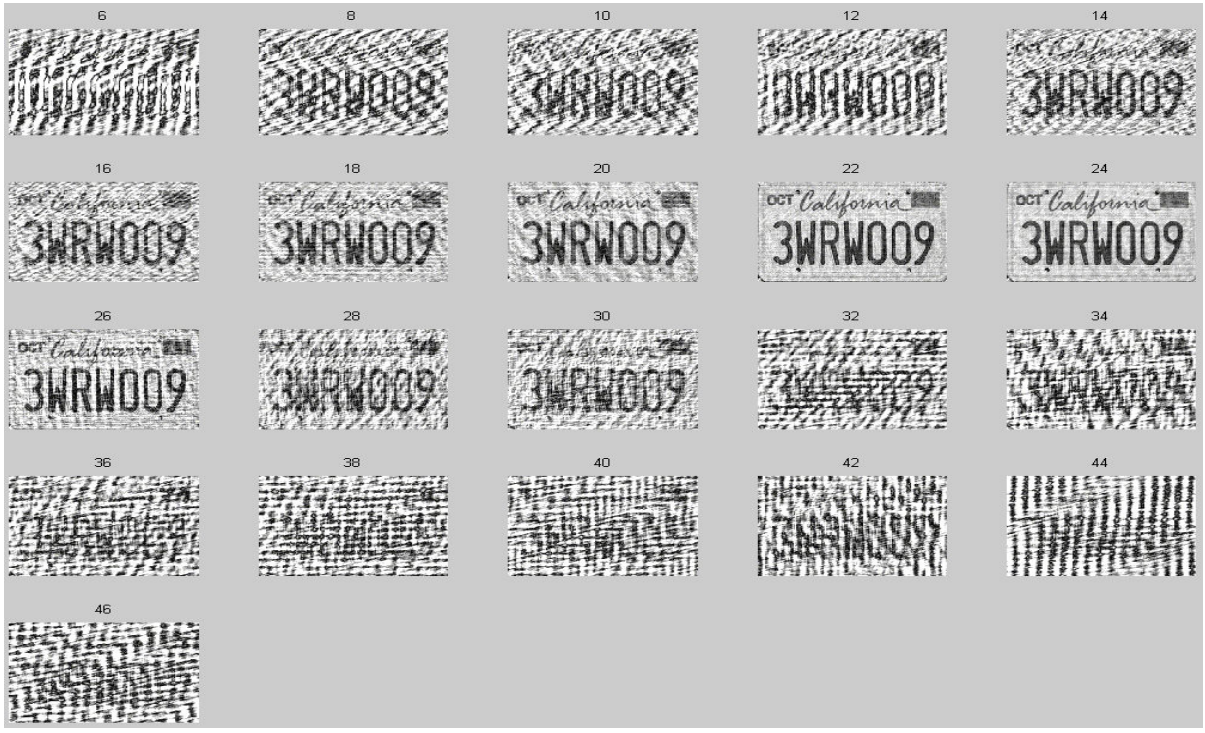

Figure 6. 2nd Iteration of User-Assisted Deblurring Algorithm

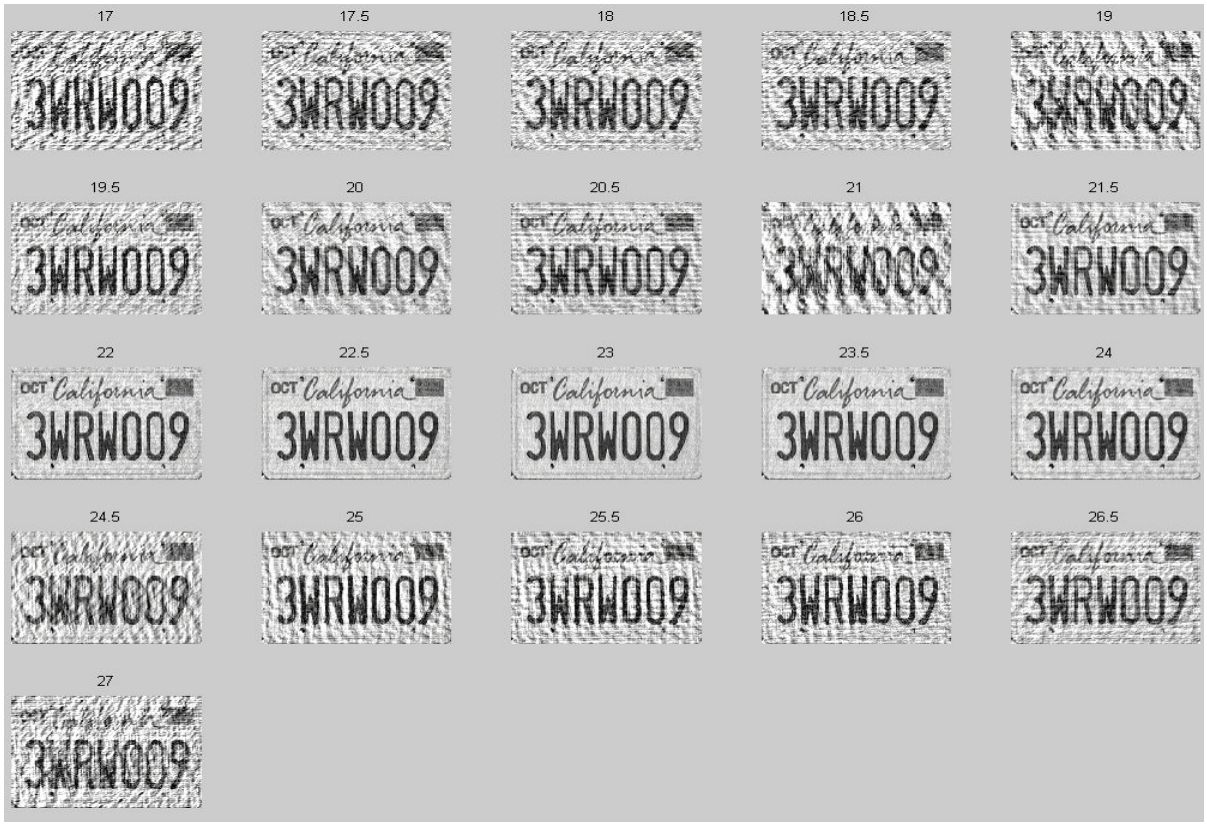

Figure 7. 3rd Iteration of User-Assisted Deblurring Algorithm

Figure 8. Original Image (left), Result of User-Assisted Deblurring Algorithm (right)

Figure 9. Effect of Noise in the Deblurring Algorithm

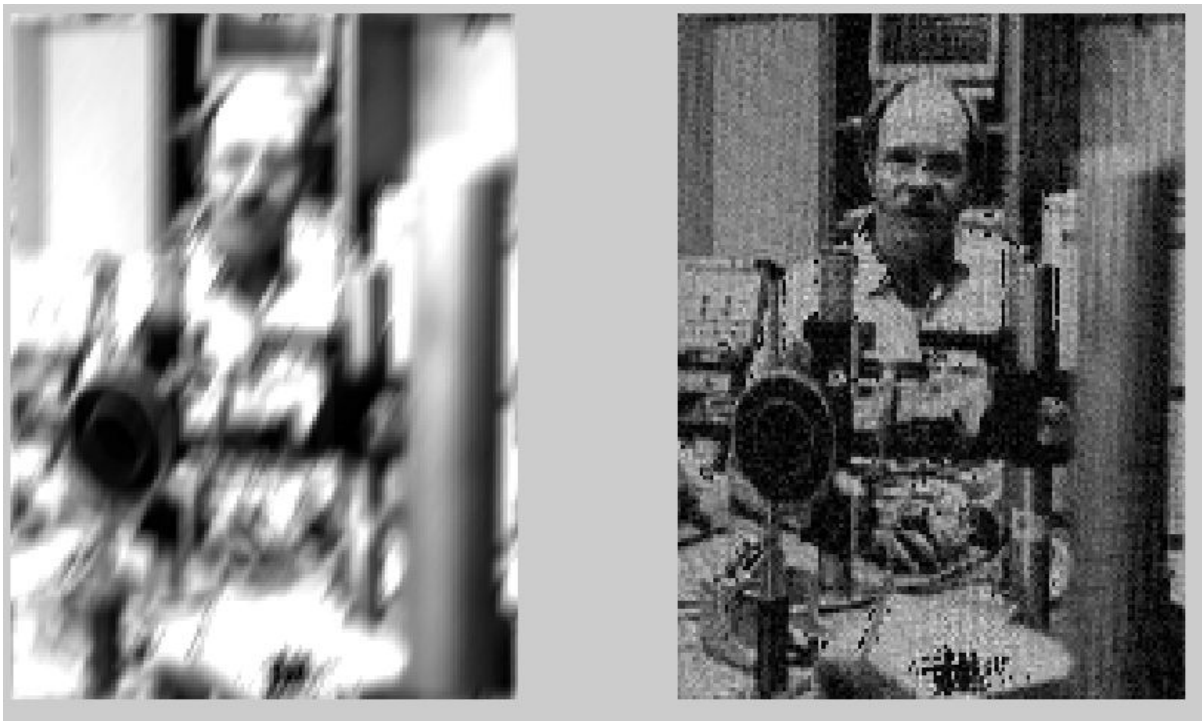

Figure 10. Unknown Blur in Image with Gaussian Noise

APPENDIX B – MATLAB SOURCE CODE

APPENDIX B – MATLAB SOURCE CODE

```
function B = unblur(A,len,theta,noofest);

B = double(A);
if len > 0
    B = uint8(Degrade(B,len,theta));
end

B2d1 = B(:,:,1);
B2d2 = B(:,:,2);
B2d3 = B(:,:,3);

esttheta1 = EstAngle(B2d1, 1, noofest);
esttheta2 = EstAngle(B2d2, 0, noofest);
esttheta3 = EstAngle(B2d3, 0, noofest);
esttheta = (esttheta1 + esttheta2 + esttheta3) / 3;

for i = 1:1:length(esttheta)
    estlength1(i) = EstLen(B2d1, esttheta(i), 1, 'working...');
    estlength2(i) = EstLen(B2d2, esttheta(i), 0, 'working...');
    estlength3(i) = EstLen(B2d3, esttheta(i), 0, 'working...');
    estlength(i) = (estlength1(i) + estlength2(i) + estlength3(i)) / 3;
end

i = 1;
input = 2;
while (input == 2)
    C = uint8(abs(Inverse(double(B),estlength(i),esttheta(i))));
    figure(1); subplot(1,1,1); image(C);
    if i < length(esttheta)
        input = menu('Is this a good image','Yes','No, look at next
image');
    else
        input = menu('Is this a good image','Yes','No, return to first
image');
        i = 0;
    end
    i = i + 1;
end

figure(1);
subplot(1,2,1);
image(A);
subplot(1,2,1);
image(B);
subplot(1,2,2);
image(C);

[m,n,l] = size(C);
C(1:1:m,1:1:n,1) = A(1:1:m,1:1:n,1);
end
```

```
function B = unblur(A,len,theta);

B = double(A);
if len > 0
    B = uint8(Degrade(B,len,theta));
end
B2d1 = B(:,:,1);

esttheta = [1:5:176];
figure(1);
for i = 1:1:length(esttheta)
    subplot(6,6,i);
    estlength = EstLen(B2d1, esttheta(i), 0, 'working...');
    C = uint8(abs(Inverse(double(B),estlength,esttheta(i))));
    image(C); axis off; title([num2str(esttheta(i)) '  '
num2str(estlength)]);
end

ui = input('Enter the best looking angle: ');
esttheta = [mod((ui-20),180):2:mod((ui+20),180)];
close(1); figure(1);
for i = 1:1:length(esttheta)
    subplot(5,5,i);
    estlength = EstLen(B2d1, esttheta(i), 0, 'working...');
    C = uint8(abs(Inverse(double(B),estlength,esttheta(i))));
    image(C); axis off; title(num2str(esttheta(i)));
end

ui = input('Enter the best looking angle: ');
esttheta = [mod((ui-5),180):0.5:mod((ui+5),180)];
close(1); figure(1);
for i = 1:1:length(esttheta)
    subplot(5,5,i);
    estlength = EstLen(B2d1, esttheta(i), 0, 'working...');
    C = uint8(abs(Inverse(double(B),estlength,esttheta(i))));
    image(C); axis off; title(num2str(esttheta(i)));
end

ui = input('Enter the best looking angle: ');
estlength = EstLen(B2d1, ui, 0, 'working...');
C = uint8(abs(Inverse(double(B),estlength,ui)));
close(1); figure(1); image(C); axis off;

figure(1);
subplot(1,2,1);
image(FSCS(B));
axis off;

subplot(1,2,2);
image(FSCS(C));
axis off;

[m,n,l] = size(C);
C(1:1:m,1:1:n,1) = A(1:1:m,1:1:n,1);
end
```

```
function [THETAS] = EstAngle(ifbl, expertstatus, noofest)
%Function to estimate blur angle
%Inputs: ifbl, expertstatus
%Returns: THETAS
%
%ifbl:  It is the input image.
%expertstatus: It decides whether to display plots and images or no
%       1 - Display plots and images
%       0 - Do not display
%handle: It is the handle to the waitbar(progress bar)
%THETAS: It is the blur angle. The angle at which the image is blurred.
It
%        is a collection of possible blur angles.
%
%Example:
%       [THETAS] = EstAngle(image, expertstatus, handle);
%       This call takes image as input and returns a group of blur
angle.

%No of steps in the algorithm
steps = 8;

%Number of estimates required
%noofest = 10;

%Preprocessing
%Performing Median Filter before restoring the blurred image
ifbl = medfilt2(abs(ifbl));

%Display input image
if expertstatus
    figure(1);
    subplot(1,2,1);
    imshow(abs(ifbl));
    title('Input image');
end

%We have to convert the image to Cepstrum domain
%This is how we represent Cepstrum Domain
%Cep(g(x,y)) = invFT{log(FT(g(x,y)))}

%Converting image from spatial domain to frequency domain
fbl = abs(fft2(double(ifbl)));

%Performing Log Transform
lg = log(1+fbl);
lgpow = abs(lg).^2;

%Converting to cepstral domain
lgcep = ifft2(lgpow);

%Finding edges in image
BW = edge(lgcep);
BW = ifftshift(BW);
```

```
%Display Edge image
if expertstatus
    subplot(1,2,2);
    imshow(abs(BW));
    title('Edge image');
end

%Calling hough transform
h = Hough(BW);
siz = size(ifbl);
rl = ceil(sqrt(siz(1)^2+siz(2)^2));

%Finding first maximum in the accumulator
maxi = 0;
theta = 0;
for i = 1:rl
    for j = 1:360
        if h(i,j)>maxi
            maxi = h(i,j);
            theta = j;
        end
    end
end

%Storing first maximum in the array
g = 1;
maxarr(g) = theta;
g = g + 1;

%Saving our original accumulator array
h2 = h;

%Iterating 10 times to find 10 highest angle values
for p = 1:(noofest-1)
    %Band Elimination the region of +5 & -5 degrees of the maximum
angle
    %If angle is between 0 and 5
    if theta<=5
        for j = 1:theta+5
            for i = 1:rl
                h2(i,j)=0;
            end
        end

        for j = 355:360
            for i = 1:rl
                h2(i,j)=0;
            end
        end

    %If angle is between 355 and 360
    elseif theta>=355
        for j = theta-5:360
            for i = 1:rl
                h2(i,j)=0;
```

```
            end
        end
        for j = 1:360-theta
            for i = 1:rl
                h2(i,j)=0;
            end
        end

    %For any other angle
    else
        for j = theta-5:theta+5
            for i = 1:rl
                h2(i,j)=0;
            end
        end
    end

    %Finding the next maximum
    maxi = 0;
    theta = 0;
    for i = 1:rl
        for j = 1:360
            if h2(i,j)>maxi
                maxi = h2(i,j);
                theta = j;
                r1 = i;
            end
        end
    end

    %Storing next maximum in the array
    maxarr(g) = theta;
    g = g + 1;
end

newarr = maxarr;
arrsiz = size(maxarr);

%Fitting it into 180 degrees
for i = 1:arrsiz(2)
    if newarr(i)>180
        newarr(i) = newarr(i)-180;
    end
end

%Possible values of THETA
THETAS = newarr;
```

```
function LEN = EstLen(ifbl, THETA, expertstatus, handle)
%Function to estimate blur length
%Inputs:  ifbl, THETA, expertstatus
%Returns: LEN
%
%ifbl:  It is the input image.
%THETA: It is the blur angle returned from angle estimation.
%expertstatus: It decides whether to display plots and images or no
%       1 - Display plots and images
%       0 - Do not display
%handle: It is the handle to the waitbar(progress bar)
%LEN:   It is the blur length. The length is the number
%       of pixels by which the image is blurred.
%
%Example:
%       LEN = EstLen(image, THETA, expertstatus);
%       This call takes image as input and returns the blur length.

%No of steps in the algorithm
steps = 8;

%Preprocessing
%Performing Median Filter before restoring the blurred image
ifbl = medfilt2(abs(ifbl));

%Display input image
if expertstatus
    figure(1);
    subplot(1,2,1);
    imshow(abs(ifbl));
    title('Input image');
end

%We have to convert the image to Cepstrum domain
%This is how we represent Cepstrum Domain
%Cep(g(x,y)) = invFT{log(FT(g(x,y)))}

%Converting to fourier domain
fin = fft2(double(ifbl));

%Performing log transform
lgfin = abs(log(1 + abs(fin)));

%Performing inverse fourier transform to get the image in Cepstrum
domain
cin = ifft2(lgfin);

%Rotating the image
cinrot = imrotate(cin, -THETA);

%Taking average of all the columns
for i=1:size(cinrot, 2)
    avg(i) = 0;
    for j=1:size(cinrot, 1)
        avg(i) = avg(i) + cinrot(j, i);
```

```
    end
    avg(i) = avg(i)/size(cinrot, 1);
end
avgr = real(avg);

%Displaying the average of all the pixels
if expertstatus
    subplot(1,2,2);
    plot(avgr); grid on;
    title('Average of pixels in cepstrum domain');
end

%Calculating the blur length using first negative value
index = 0;
for i = 1:round(size(avg,2)),
    if real(avg(i))<0,
        index = i;
        break;
    end
end

%If Zero Crossing found then return it as the blur length
if index~=0,
    LEN = index;
else
    %If Zero Crossing not found then find the lowest peak
    %Calculating the blur length using Lowest Peak
    index = 1;
    startval = avg(index);
    for i = 1 : round(size(avg, 2)/2),
        if startval>avg(i),
            startval = avg(i);
            index = i;
        end
    end

    LEN = index;
end
```

```
function ifbl = degrade(im, LEN, THETA)
%Function to degrade image
%Inputs: im, LEN, THETA, Noise type, M & V
%Returns: ifbl
%
%Description:
%im:    It is the input image
%LEN:   It is the no. of pixels by which we can blur the image
%       eg.: (1 - 100)  (Less Blur - More blur)
%THETA: It is the angle at which we have to blur the image
%       eg.: (0 - 180)
%noisetype: It is the type of noise we can consider
%     We are considering 'salt & pepper', 'gaussian', 'poisson' &
'speckle'
%M:   This is
%     Mean when noisetype is 'gaussian' &
%     Density when noisetype is 'salt & pepper'.
%     Default is 0 for 'gaussian'
%     Default is 0.05 for 'salt & pepper'
%V:   This is the variance to the image
%     Default is 0.01
%ifbl: This is the blurred image in spatial domain
%
%Example:
%       ifbl = degrade(im, 30, 20, 'salt & pepper')
%       This call blurs the image with length 30 and angle 20.
%       'salt & pepper' noise is added to the image.

%Converting to double
imd = im2double(im);

%Converting the image to frequency domain
imf = fft2(imd);

%Create PSF of degradation
PSF = fspecial('motion',LEN,THETA);

%Convert psf to otf of desired size
%OTF is Optical Transfer Function
%fbl is blurred image in frequency domain
OTF = psf2otf(PSF,size(im));

%Blurring the image
fbl = OTF.*imf;

%Converting back to spatial domain
ifbl = abs(ifft2(fbl));


end
```

```
function resim = Inverse(ifbl, LEN, THETA)
%Function to restore the image using Inverse Filter
%Inputs: ifbl, LEN, THETA.
%Returns: resim.
%
%ifbl:  It is the input image.
%THETA: It is the blur angle. The angle at which the image is blurred.
%LEN:   It is the blur length. The length is the number
%       of pixels by which the image is blurred.
%handle:It is the handle to the waitbar(progress bar).
%resim: It is the restored image.
%
%Example:
%       resim = Inverse(image, LEN, THETA);
%       This call takes image, blur length & blur angle as input
%       and returns the restored image.

%No of steps in the algorithm
steps = 6;
handle = 'Inverse function running...';

%Converting to frequency domain
fbl = fft2(ifbl);

%Create PSF of degradation
PSF = fspecial('motion',LEN,THETA);

%Convert psf to otf of desired size
%OTF is Optical Transfer Function
OTF = psf2otf(PSF, size(fbl));

%To avoid divide by zero error
min = 0.01;
for i = 1:size(OTF, 1)
    for j = 1:size(OTF, 2)
        if ((abs(OTF(i, j, 1)) < min) | (abs(OTF(i, j, 2)) < min) |
(abs(OTF(i, j, 3)) < min))
            OTF(i, j, 1) = 1;
            OTF(i, j, 2) = 1;
            OTF(i, j, 3) = 1;
        end
    end
end

%Restoring the image using Inverse Filter
fdebl = fbl./OTF;

%Converting back to spatial domain using IFFT
resim = ifft2(fdebl);

%figure(2);
image(fftshift(uint8(FSCS(log(abs(fft2(double(imread('california.bmp'))
)))))));
%title('original');
%figure(3); image(fftshift(uint8(FSCS(log(abs(fbl))))));
```

```
%title('blurred');
%figure(4); image(fftshift(uint8(FSCS(log(abs(fdebl))))));
%title('deblurred');
%figure(5); image(fftshift(uint8(FSCS(abs(OTF)))));
%title('OTF');
end
```