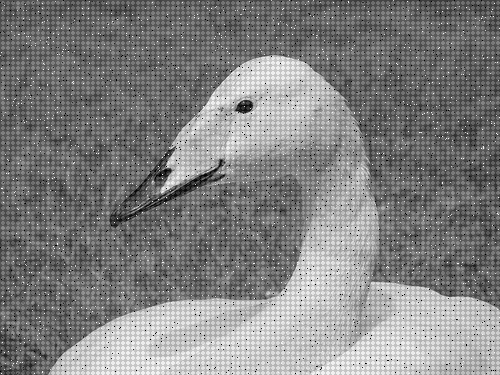
# Image Analysis - Image Enhancement

**Abstract – This report details the analysis of noise removal and colour enhancement using image processing techniques. Different techniques are applied and analysed to deduce a combination of image processing filters to enhance the target image swanNoise.bmp. Alternative methods of these techniques are also examined and discussed.**

## Introduction

**Figure.1** displays an original image of a swan and **Figure.2** displays a noisy swan image. It can be seen that the image has been corrupted with periodic (structured) and random (salt and pepper) noise. Methods and techniques of noise filtering are implemented and tested throughout this report to observe how data such as noise can be isolated and extracted from image. The methods results can be compared to deduce an efficient noise filter for the swan image.

** Figure.1** swanOriginal.jpg  **Figure.2** swanNoise.jpg

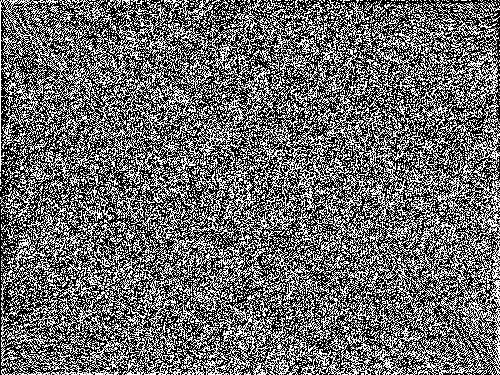


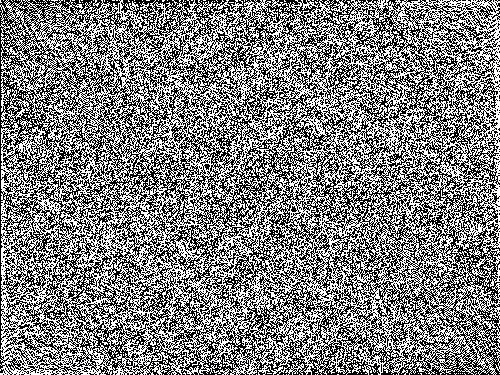
## Methodology and Results

The noisy swan image will be subject to an averaging filter to remove random noise and Fourier transformation to remove periodic noise. Fourier transformation will allow us to transfer the representation of an image from its spatial domain to its frequency domain. This provides more efficient methods for isolating and manipulating particular characteristics of the data.

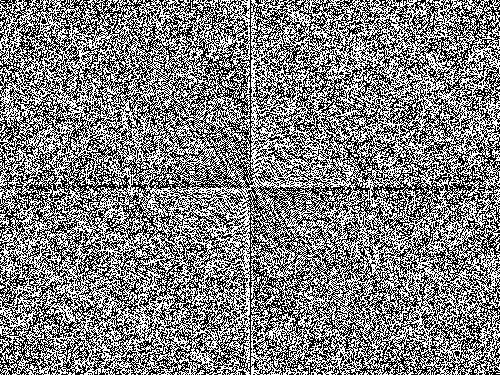
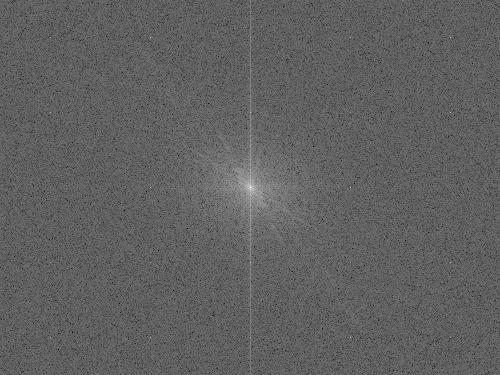
## Fourier Transformation

The original Fourier Transformation **Figure.3** is calculated by performing fast fourier transformation on the original image. The fourier representation of an image contains real and imaginary data. These types of data can be isolated into the magnitude and phase of the image. The angles of the fourier data are then computed to provide us with **the** phase displayed in **Figure.4**.

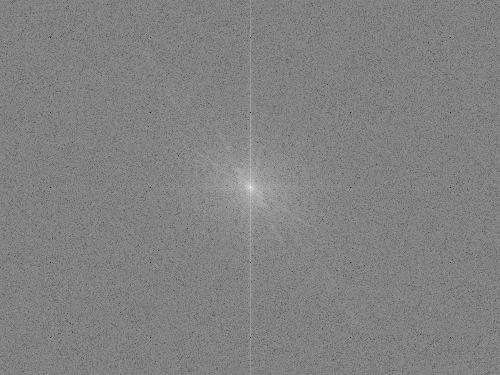
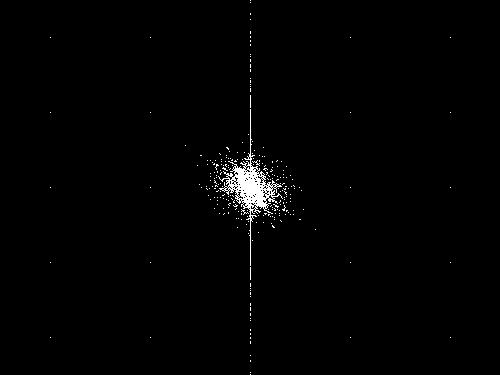
** Figure.3** Original FFT **Figure.4** Phase

****

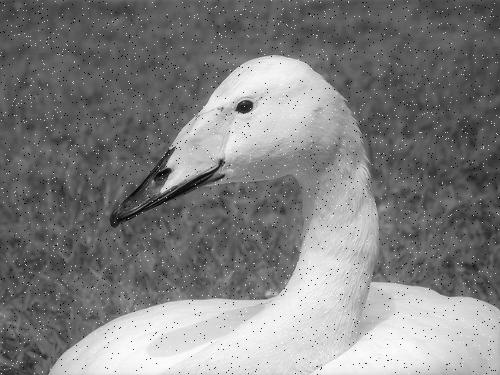
Zero-frequency components are then shifted to the centre by swapping quarters of the image diagonally. **Figure.5** shows the result of the shifted phase data. The magnitude of the image is found by calculating the logarithmic absolute values of the shifted phase displayed in **Figure.6**.

** Figure.5** Shifted Phase **Figure.6** Magnitude

Converting the image to binary allows us to see the structured noise much clearer although further data is lost such as greyscale intensities. This does not concern us as the binary magnitude is literally a filter to reference the location of the white pixels that are manipulated in the original magnitude.

**** **Figure.7** Binary Magnitude **Figure.8** Filtered Magnitude

A simple program has been written in java **Appendix.1** to iterate through a matrix and find the locations of white pixels. There is logic to disregard any white pixels found within a defined rectangle in the matrix. This rectangle resides in the centre of the matrix preserve the highest frequency data. The located pixel value coordinates from **Figure.7** are then set to zero (black) in original magnitude **Figure.6** to derive a filtered magnitude displayed in **Figure.8.** To compute the periodically filtered image the filtered magnitude must then be multiplied by the shifted phase. Inverse fourier transformation is performed on the result of the previous step to deliver a swan image without periodic noise displayed in **Figure.9**.

 **Figure.9** Periodic filtered image **Figure.10** Random noise removal (median filter)

## Averaging filters

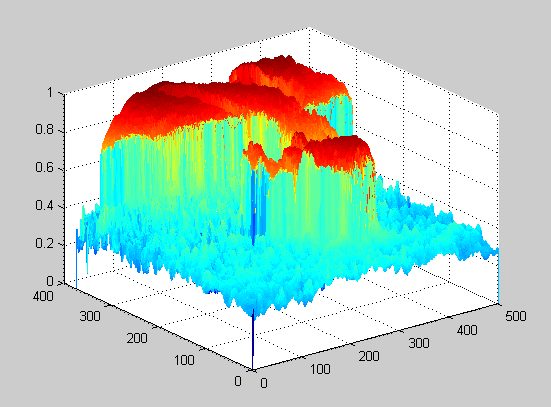
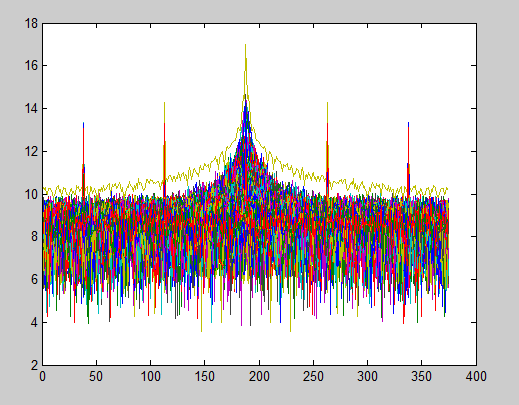
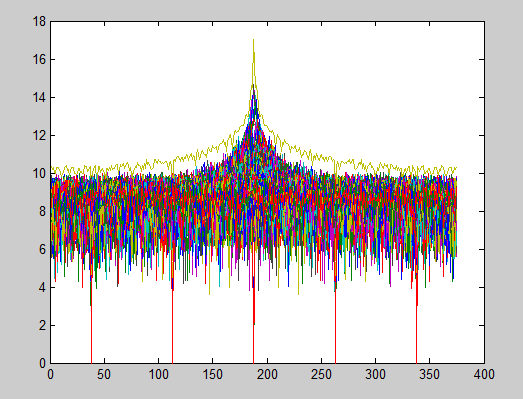
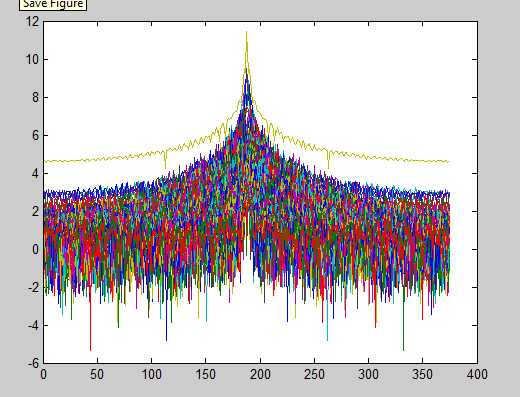
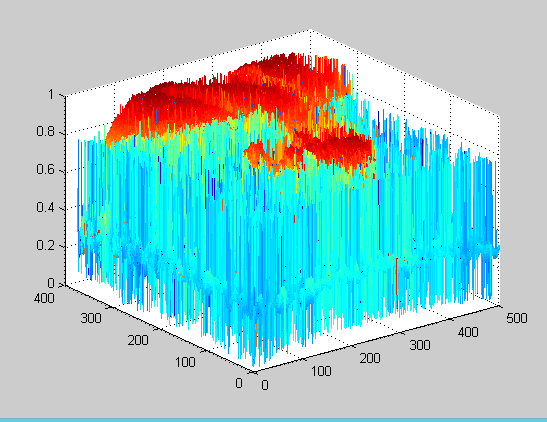
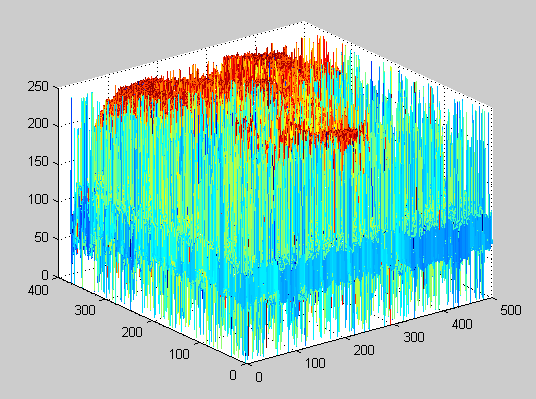
The averaging filters calculate pixel averages within a defined neighbourhood of pixels. A 3x3 kernel is used as a filter to average the pixels within the filter. Values at the edge of the image are padded with zeros. As a result of this we would expect to see all values in the neighbourhood set to their average which should result in smoothing in the image and the removal of the salt and pepper noise. **Figure.10** displays the results of a median filter being applied to the swan image with its periodic noise removed. It can be seen that the image has blurred slightly as a result of the median filters pixel averaging. The filtered image is then sharpened using a Matlab function ‘imsharpen’ which brings some resolution back to the image as seen in **Figure.11**.

**Figure.11** Sharp Median Filtered Image (medfilt2) **Figure.12** Sharp Adaptive Filtered (Wiener)

Other methods of 2-D filters are explored such adaptive filtering. Adaptive filters learn as they go and manipulate filter parameters over time [1]. The adaptive filter **Figure.12** visually looks as though it loses more image detail than the median filter.

To visualize the spread of intensity within the 2-D fourier data **Figure.13** illustrates pixel intensities at each of the three stages of enhancement. Step 1: Original image, step 2: periodic noise removed and step 3: periodic and random noise removed. The spike displayed in the second row of the graphs represents the centralized data from **Figure.6** at each stage of the noise removal**.**

**Figure.13** 3 steps of enhancement. (Original image, periodic removed, periodic & random removed)



**Appendix.2** illustrates the noise removal code used to filter the swan image

## Alternative attempt of Noise removal

Another method of noise filtering was implemented using **Appendix.3** but did not prove effective. This was filtering used a Matlab function called ‘conv’ which essentially calculates the convolution of two given matrices. In the resulting image **Figure.14** there is a mild black line around the outside of the image which could be that the edges have not been padded before the kernel iterated through the image. The image is also highly distorted and a bad representation of an enhancement.

**Figure.14** Alternative method of noise removal

## Colour Enhancement

## HSI Colour model manipulation

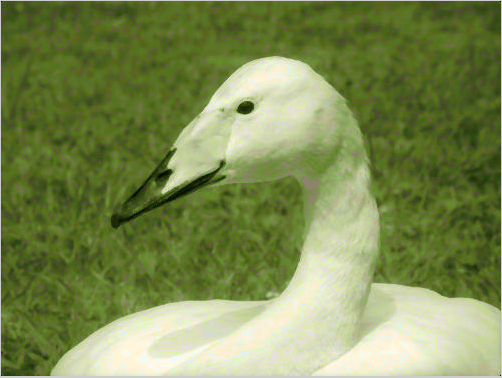
An attempt to manipulate the intensity of the HSI model was carried out with the following code **Appendix.4.** The logic here is setting the intensity value of a pixel read from the greyscale image to the green channel of the HSI model. The results displayed in **Figure.15** show that this is not a very good solution as the image is overwhelmed with green and much of the images details have been lost. For a better result the colour intensities could be finer tuned.

 **Figure.15** HSI colour Enhancement **Figure.16** YCBCR Enhance with swan2.jpg



## YCBCR Colour Enhancement

The filtered swan image has been put through an YCBCR enhancement filter at an attempt to restore original colours to the image. As the subject image of this project is originally greyscale recovering a colour map is not possible. This enhancement technique works by calculating the CB and CR (Chrominance) and luminance of an RGB colour image similar to the target enhancement image **Figure.16**. It then adjusts the chrominance and luminance of the target image to match the sources. The implementation of the filter is very slow and inefficient but delivers an effect result **Figure.16**. The code for this filter can be found **Appendix.5.** The grey filtered swan image **Figure.20** was then YCBCR enhanced with Figure.19 **swan3.jpg**. This looks like even better result with small colouring on the beak and slightly greener grass.

 **Figure.17** swan2.jpg **Figure.18** YCBCR Enhance with swan3.jpg

**Figure.19** swan3.jpg



## Discussion and Conclusion

As it can be seen in the **Figure.18** the combination of filters applied to remove noise and restore colouration to the image are the most efficient of the methods observed whilst retaining the most detail. Visually the quality of the filtered image can be seen against the original in **Figure.1.**

The matrix of the 3x3 kernel used for the median filtering is an effective size for the amount of random noise present. If a median filter was to be applied to an image with more noise or larger dimensions then a larger filter would be more effective. A 4x4 filter was tested but resulted in higher rates of blurring and data loss in the image. The filter used in this image project could be improved by locating the random noise pixels which display as high intensities of black and white and targeting only those areas with an averaging filter. The median filter delivered less data loss than the adaptive filter in the context of filtering the swan image.

The fourier domain is an efficient way of removing noise from an image. The manual implementation of the ideal low-pass filter like the one created in this project delivers effective results. The logic behind the noise removal filter developed in this project performs calculations on a 2-D matrix representation. Other automated imaging filters operating in the fourier domain represent and calculate their data slightly differently as they view the frequency wave opposed to a matrix representation of the frequencies in **Figure.6.** Essentially the logic is the same where frequencies within a given threshold will be nullified or enhanced to compute a desired effect.

The YCBCR Colour Enhancement filter derives a YCBCR colour model from an RGB colour model of a similar looking image. It currently takes minutes to run but could be dramatically be reduced in run time by pre-allocating memory to store and compute the calculated channels opposed to gathering this data every iteration of the loop. Further improvements to the enhancement of the image could segmenting the image and create colour masks. Similar methods of YCBCR colour enhancement encompass performing histogram matching of the plotted luminance and Chrominance values [2].

## References

[1] Mathworks, Adaptive Filters, available at: <http://uk.mathworks.com/help/dsp/ug/overview-of-adaptive-filters-and-applications.html#f1-12973>

[2] Journal of Theoretical and applied information technology, Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.401.9165&rep=rep1&type=pdf>

[3] Mathworks, Noise Removal, Available: <http://uk.mathworks.com/help/images/noise-removal.html>

[4] Cliffsview, Available: <https://cliffsview.files.wordpress.com/2014/12/whooper-swans-greglag-geese-neighbours-for-christmas-xs6731.jpg>

[5] Mathworks, Available: <http://uk.mathworks.com/matlabcentral/answers/81048-mse-mean-square-error>

[6] Mathworks, Available: <http://in.mathworks.com/matlabcentral/fileexchange/8214-gray-image-to-color-image-conversion>

[7] Mathworks, rgb2ycbcr, Available: <http://uk.mathworks.com/help/images/ref/rgb2ycbcr.html>

## Appendices

**Appendix.1 PixelDetector.java**

public class Main {

public static void main(String[] args) throws IOException {

// read in image

String path = "C:\\Users\\user\\Documents\\Uni Work\\Uni-Work-2015\\Modules\\Image Processing\\Coursework 1\\Course Work Images\\fftMagnitude.jpg";

BufferedImage image = ImageIO.read(new File(path));

for(int r=0; r<image.getWidth(); r++)

{

for(int c=0; c<image.getHeight(); c++)

{

// get 3 channel color from each pixel

Color color = new Color(image.getRGB(r, c));

// get average color of each pixel

int avgColor = (color.getRed() + color.getBlue() + color.getGreen())/3;

//logic to disregard white pixel count in defined coordinates

if(c < 120 || c > 340 )

{

if(avgColor == 255)

{

// print detected pixels to console

System.out.println(r + ", " + c + " = " + color);

}

}

}

}

}

}

**Appendix.2** **NoiseRemoval.m**

%%% Periodic and Random noise removal %%%

% image filepath

path = '\\ndrive\xw009807\.do\_not\_delete\desktop.xp\IA assignment\Images\swanNoise.bmp';

path2 = '\\ndrive\xw009807\.do\_not\_delete\desktop.xp\IA assignment\Images\swanOriginal.bmp';

% read images

NI = imread(path); % noisy image

OI = imread(path2); % original image

% fourier transform

originalfft = fft2(NI);

% shift fourier data so that lowest frequencies are centered

shiftedfft = fftshift(originalfft);

% calculate fourier magnitude

fftMagnitude = log(abs(shiftedfft));

fftMagnitude = fftMagnitude - min(fftMagnitude(:));

fftMagnitude = fftMagnitude ./ max(fftMagnitude(:));

%binaries image

binfftMagnitude = im2bw(fftMagnitude, 0.5);

% calculate phase of Fourier

fftPhase = angle(originalfft);

% pixels located by pixelDetector.java

% set located pixels to zero

fftMagnitude(38,51) = 0 ;

fftMagnitude(38,151) = 0 ;

fftMagnitude(38,251) = 0 ;

fftMagnitude(38,351) = 0 ;

fftMagnitude(38,451) = 0 ;

fftMagnitude(113,51) = 0 ;

fftMagnitude(113,151) = 0 ;

fftMagnitude(113,251) = 0 ;

fftMagnitude(113,351) = 0 ;

fftMagnitude(113,451) = 0 ;

fftMagnitude(188,51) = 0 ;

fftMagnitude(188,151) = 0 ;

fftMagnitude(188,351) = 0 ;

fftMagnitude(188,451) = 0 ;

fftMagnitude(263,51) = 0 ;

fftMagnitude(263,151) = 0 ;

fftMagnitude(263,251) = 0 ;

fftMagnitude(263,351) = 0 ;

fftMagnitude(263,451) = 0 ;

fftMagnitude(338,51) = 0 ;

fftMagnitude(338,151) = 0 ;

fftMagnitude(338,251) = 0 ;

fftMagnitude(338,351) = 0 ;

fftMagnitude(338,451) = 0 ;

% reverse fourier transformation

inverseFFT = mat2gray(abs(ifft2(fftMagnitude .\* shiftedfft)));

% apply median filter to remove random noise

filteredImage = medfilt2(inverseFFT, [3 3]);

% sharpen image

imsharpen(filteredImage);

% display filtered image

imshow(filteredImage);

%calculate mean square error of final image

squaredErrorImage = (double(OI) - (double(NI)) .^ 2);

[rows,columns] = size(OI);

mse = sum(sum(squaredErrorImage)) / (rows \* columns);

**Appendix.3 NoiseRemoval2.m**

I = imread('\\ndrive\xw009807\.do\_not\_delete\desktop.xp\IA assignment\Images\swanNoise.bmp');

originalfft = fft2(I);

shiftedfft = fftshift(originalfft);

rectWidth = 15;

rectHeight = 10;

kernel = ones(rectWidth, rectHeight) / (rectHeight \* rectWidth);

filteredImage = conv2(double(I), kernel, 'same');

filteredImage = filteredImage - min(filteredImage(:));

filteredImage = filteredImage ./ max(filteredImage(:));

imsharpen(filteredImage);

imshow(filteredImage);

imwrite(filteredImage,'noiseRemovalMethid2.jpg');

**Appendix .4 HSIColorEnhancement.m**

% HIS color enhancement

% image filepaths

path = '\\ndrive.rdg.ac.uk\xw009807\My Documents\MATLAB\noiseFilteredImage.jpg';

% read RGB and greyscale swan images

I = imread(path);

% get image dimensions

[M,N] = size(I);

% create 3-channel matrix to hold rgb values

rgbimg = zeros(M,N,3);

rgbimg(:,:,1) = rgbimg(:,:,1);

rgbimg(:,:,2) = I;

rgbimg(:,:,3) = rgbimg(:,:,1);

HSII = rgb2hsv(rgbimg/255);

imshow(HSII);

imwrite(HSII,'HSIimage.jpg');

**Appendix.5 YCBCRColorEnhancemnt.m**

%% YCBCR Color enhancement %%

% filepaths

img1 = '\\ndrive.rdg.ac.uk\xw009807\My Documents\MATLAB\NoiseFilteredimage.jpg'; % Source image

img2 = '\\ndrive.rdg.ac.uk\xw009807\My Documents\MATLAB\swan3.jpg'; % Target Image

% read images

targetImage = imread(img1);

sourceImage = imread(img2);

% get image dimensions

[sourceX sourceY sourceZ] = size(targetImage);

[targetX targetY targetZ] = size(sourceImage);

targetImage(:,:,2)=targetImage(:,:,1);

targetImage(:,:,3)=targetImage(:,:,1);

% convert to ycbcr color space

space1 = rgb2ycbcr(sourceImage);

space2 = rgb2ycbcr(targetImage);

% convert data to double and store in matrix for source and target image

matSource = double(space1(:,:,1));

matTarget = double(space2(:,:,1));

% get max min of source and target

mat1=max(max(matSource));

mat2=min(min(matSource));

mat3=max(max(matTarget));

mat4=min(min(matTarget));

dim1=mat1-mat2;

dim2=mat3-mat4;

% Normalization

dx1 = matSource;

dx2 = matTarget;

dx1 = (dx1\*255)/(255-dim1);

dx2 = (dx2\*255)/(255-dim2);

[mx,my,mz] = size(dx2);

% Pre-define memory in temp matrix

%Luminance Comparison

disp('Please wait..................');

for i=1:mx

for j=1:my

iy=dx2(i,j);

tmp=abs(dx1-iy);

ck=min(min(tmp));

[r,c] = find(tmp==ck);

ck=isempty(r);

if (ck~=1)

nimage(i,j,2) = space1(r(1),c(1),2);

nimage(i,j,3) = space1(r(1),c(1),3);

nimage(i,j,1) = space2(i,j,1);

end

end

end

rslt=ycbcr2rgb(nimage);

% display original and esult imagesr

figure,imshow(uint8(targetImage));

figure,imshow(uint8(rslt));

R=uint8(rslt);

imwrite(R,'ColourEhancedimage.jpg');