Digital Image Processing Laboratory

2020-2021

المرحلة الثالثة / الفصل الدراسي الثاني

اساتذة المادة:

أ.م.د. ناصر حسين سلمان

م.د. سهيلة نجم محمد

م.م. هدى مصطفى رضا

م.م. فاتن صادق خزعل

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 1: Image Basics

1. Image Definition

An ***image*** is a two-dimensional array (a matrix), of picture elements (called pixels) arranged in columns and rows as shown in Figure (1).

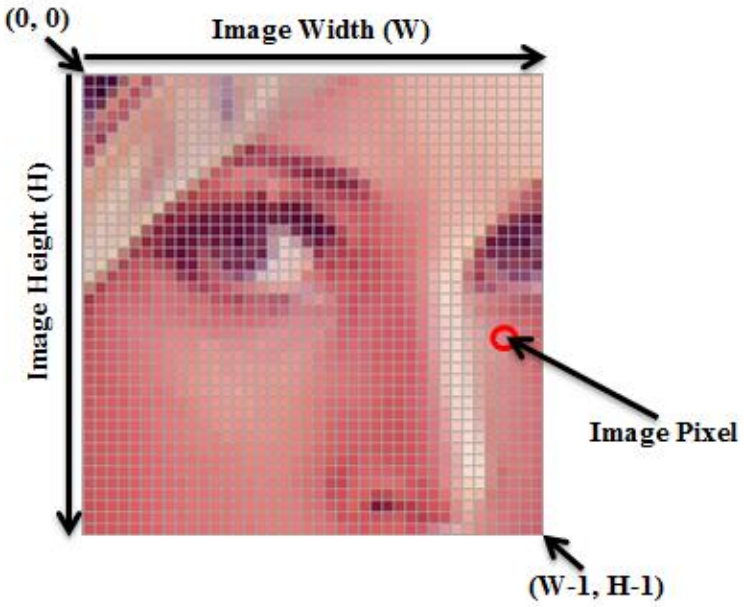
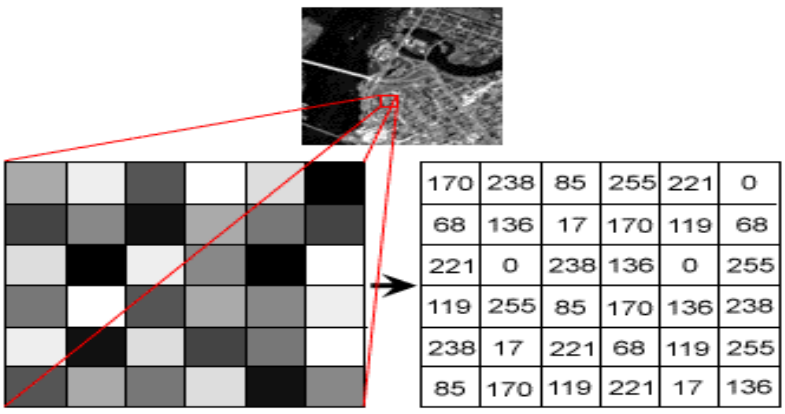


Figure (1): An image example

2. Grey Scale Image

In "*grey scale*" image, each element has an assigned intensity with 8-bit color depth that ranges from 0 to 255. Figure (2) shows an example for grey scale image.



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Figure (2): Grey scale image example

3. True Color Image

A "*true color*" image uses 24-bit color depth to display an RGB image. The RGB model defines a color by giving the intensity level of red, green and blue light that mix together to create a pixel on the display. With most of today's displays, the intensity of each color can vary from 0 to 255, which gives 16,777,216 different colors. Table (1) shows some example colors and their red, green and blue intensity values. Figure (3) shows an example for a true color image with its RGB components.

Table (1): Some example colors and their red, green and blue intensity values

Color	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Yellow	255	255	0
Cyan	0	255	255
Magenta	255	0	255
White	255	255	255
Black	0	0	0

Original image **Red channel** **Green channel** **Blue channel**

Figure (3): True color image with its RGB components

4. Binary Image

Binary images are images whose pixels have only two possible intensity values. They are normally displayed as *black* and *white*. Numerically, the two values are often 0 for black, and either 1 or 255 for white. Binary images are often produced by thresholding a grey scale or color

image, in order to separate an object in the image from the background. The color of the object (usually white) is referred to as the foreground color. The rest (usually black) is referred to as the background color. Figure (4) shows an example for a binary image.

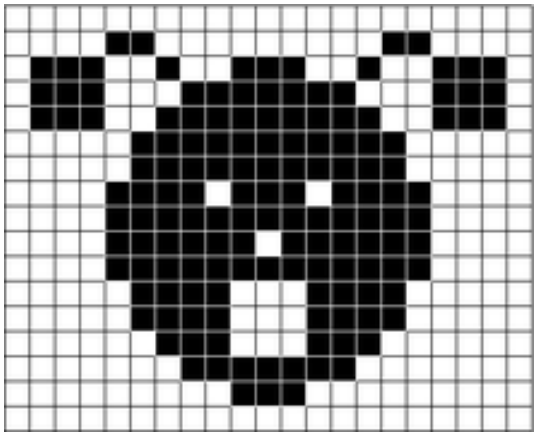


Figure (4): Binary image example

5. Lab Experiments

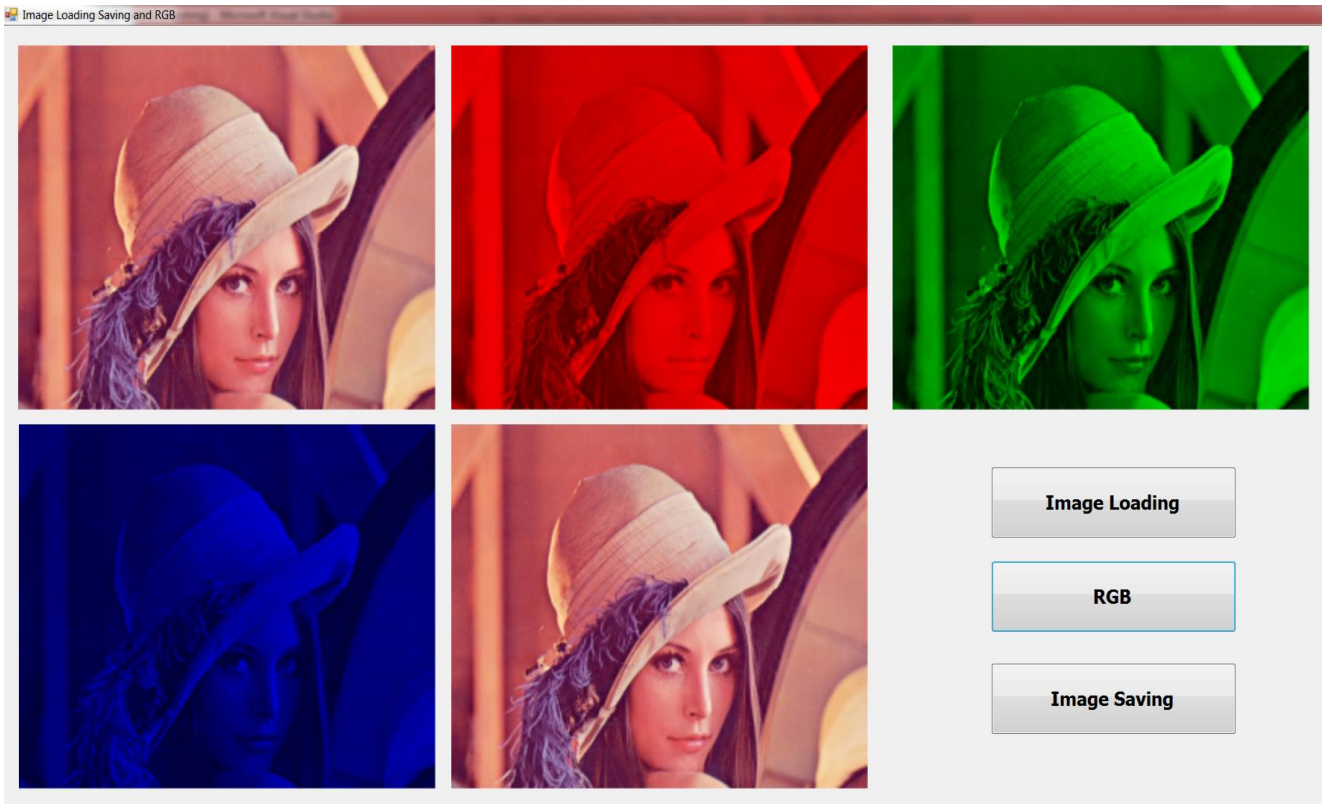
- A true color image loading.
- RGB color bands decomposition.

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 1 : Image Loading, Saving and RGB Decomposition

Design:



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3, bmp4;
        int w, h;
        byte[,] red;
        byte[,] green;
        byte[,] blue;

        public Form1()
        {
```

```
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
           // openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            saveFileDialog1.Filter = "Bmp Images |*.bmp";
            saveFileDialog1.ShowDialog();
            bmp.Save(saveFileDialog1.FileName);
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;
            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];


            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);
            bmp4 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));

                    bmp4.SetPixel(i, j, Color.FromArgb(red[i, j], green[i, j], blue[i,
                    j]));


                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;
            pictureBox5.Image = bmp4;
        }
    }
}
```

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 2: Image Manipulations

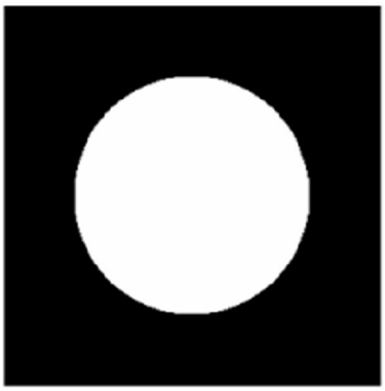
- The image of a ramp (256×256):

$$\mathbf{A} = \left[\begin{array}{cccc} 0 & 1 & 2 & \ldots & 255 \\ 0 & 1 & 2 & \ldots & 255 \\ \vdots & & & & \\ 0 & 1 & 2 & \ldots & 255 \end{array}\right] \Bigg\} 256 \text{ rows}$$

```
>> for i = 1 : 256
     for j = 1 : 256
       A(i, j) = j − 1;
       end
     end
>> image(A);
>> colormap(gray(256));
>> axis('image');
```

- The image of a circle (256×256) of radius 80 pixels centered at $(128, 128)$:

$$B(i,j) = \begin{cases} 255 & \text{if } \sqrt{(i-128)^2 + (j-128)^2} < 80 \\ 0 & \text{otherwise} \end{cases}$$



```
>> for i = 1 : 256
     for j = 1 : 256
       dist = ((i - 128)^2 + (j - 128)^2)^(.5);
       if (dist < 80)
         B(i, j) = 255;
       else
         B(i, j) = 0;
       end
     end
   end
>> image(B);
>> colormap(gray(256));
>> axis('image');
```

- The image of a "graded" circle (256×256):

$$C(i,j) = A(i,j) \times B(i,j)/255$$

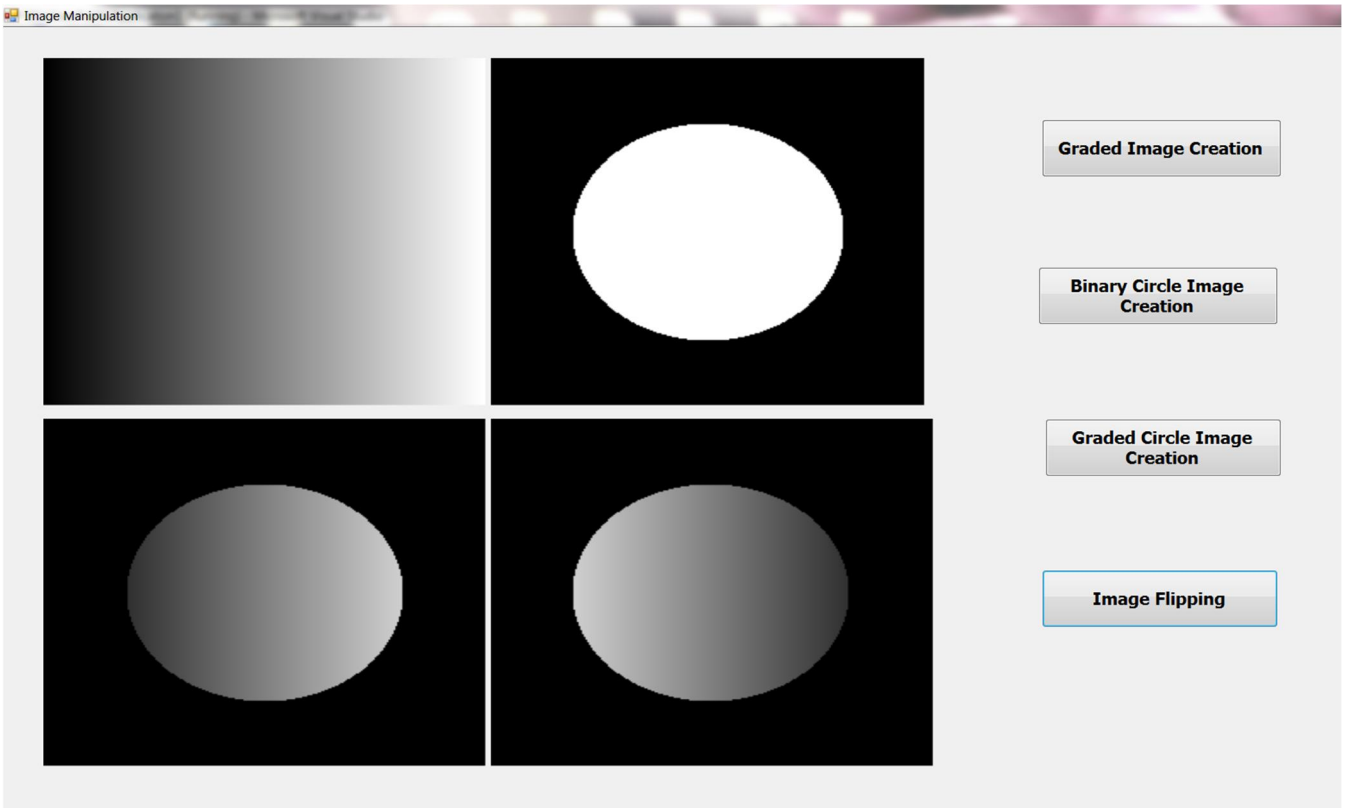
```
>> for i = 1 : 256
     for j = 1 : 256
       C(i, j) = A(i, j) * B(i, j)/255;
       end
     end
>> image(C);
>> colormap(gray(256));
>> axis('image');
```

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 2 : Image Manipulations

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        byte[,] A,B,C,F;
        int W=256, H=256;
        Bitmap bmp;

        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {

             A = new byte[W, H];
             bmp = new Bitmap(W, H);

             for (int i = 0; i < W; i++)
             {
                 for (int j = 0; j < H; j++)
                 {
                     A[i, j] = (byte)(i);
                     bmp.SetPixel(i, j,Color.FromArgb ( A[i, j], A[i, j], A[i, j]));
                 }
             }
             pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {

            B = new byte[W, H];
            bmp = new Bitmap(W, H);

            for (int i = 0; i < W; i++)
            {
                for (int j = 0; j < H; j++)
                {
                    double dis = Math.Sqrt(((i - 128) * (i - 128)) + ((j - 128) * (j -
128)));
                    if (dis < 80)
                        B[i, j] = 255;
                    else B[i, j] = 0;
                    bmp.SetPixel(i, j, Color.FromArgb(B[i, j], B[i, j], B[i, j]));
                }
            }
            pictureBox2.Image = bmp;
        }

        private void button4_Click(object sender, EventArgs e)
        {

            C = new byte[W, H];
            bmp = new Bitmap(W, H);

            for (int i = 0; i < W; i++)
            {
                for (int j = 0; j < H; j++)
                {
                    C[i, j] = (byte)(A[i,j]*(B[i,j]/255.0));
                    bmp.SetPixel(i, j, Color.FromArgb(C[i, j], C[i, j], C[i, j]));
                }
            }
            pictureBox3.Image = bmp;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            F = new byte[W, H];
            bmp = new Bitmap(W, H);

            for (int i = 0; i < W; i++)
            {
                for (int j = 0; j < H; j++)
```

```
                {
                    F[W-1-i, j] = C[i,j];
                }
            }

            for (int i = 0; i < W; i++)
            {
                for (int j = 0; j < H; j++)
                {

                     bmp.SetPixel( i, j, Color.FromArgb(F[ i, j], F[i, j], F[ i, j]));
                }
            }
            pictureBox4.Image = bmp;

         // Lab. Task: other types of flipping

        }
    }
}
```

Lab. 3: Grey Scale Conversion, Binarization, Inversion

1. Grey Scale Conversion

A grey scale (or gray level) image is simply one in which the only colors are shades of grey. In fact a `*grey*' color is one in which the red, green and blue components all have equal intensity in RGB space, and so it is only necessary to specify a single intensity value for each pixel, as opposed to the three intensities needed to specify each pixel in a full color image.

There are different methods to convert a true color image into grey scale. Followings are two methods which are commonly used:

i. Average Method

This method is the simplest and fast method. It is working by taking the average of RGB color bands for each pixel. The averaging is performed using the following equation:

Grey = (R + G + B) / 3

However, this method is taking 33% of each color band but in fact the three color bands have three different wavelengths and have their own contribution in the formation of image.

ii. Weighted Method

This method bypasses the problem of averaging method. The green color gives more soothing effect to the eyes than the two other bands while blue color is the less informatics one. So, in weighted method, Red channel has contributed 30%, Green channel has contributed 59% and Blue channel has contributed 11% as in the following equation:

Grey = 0.3 R + 0.59 G + 0.11 B

However, in comparison to the result of average method, weighted method gives a brighter image. Figure (1) shows result of applying these two methods on a true color image.

Original image **Averaging method result** **Weighted method result**

Figure (1): Result of applying the two grey scale conversation methods on a true color image

2. Image Binarization

A binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white. To convert an image into binary one, *thresholding* method is used.

The simplest thresholding method replaces each pixel in an image with a black pixel, if the image intensity I is less than some fixed constant T (that is, I<T), or a white pixel if the image intensity is greater than that constant. In other words:

$$\boldsymbol{Binary}(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} \boldsymbol{0} & \boldsymbol{if\ I}(\boldsymbol{x},\boldsymbol{y}) < \boldsymbol{T} \\ \boldsymbol{1} & \boldsymbol{otherwise} \end{cases}$$

Figure (2) shows result of applying thresholding operation on a grey scale image with T=100.

Grey scale image

Binarization result

Figure (2): Image binarization result

3. Color Inversion

In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image, in mathematic form it can be given as:

$$V_{Inv} = (L - 1) - V$$

For the grey scale image where each pixel can be represented by 8 bits (256 grey shades ranged from 0 to 255), the inversion equation is:

$$V_{Inv} = 255 - V$$

Where in the resulted image, the lighter pixels become dark and the darker picture becomes light as shown in Figure (3).

Original grey scale image

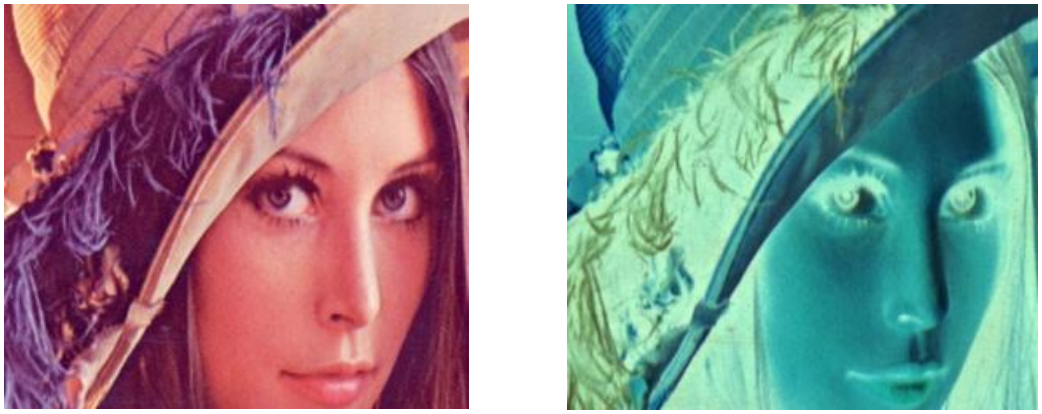
Color inversion result

Figure (3): A grey scale image inversion

For a true color image, negative transformation is applied for each color band individually as following:

$$R_{Inv} = 255 - R$$
$$G_{Inv} = 255 - G$$
$$B_{Inv} = 255 - B$$

Figure (4) shows result of applying color inversion on a true color image.



Original true color image **color inversion result**

Figure (4): A true color image inversion result

If the image is a binary one then inversion transformation will convert white pixel into black and vice versa. Figure (5) shows result of applying color inversion on a binary image.

Original binary image **Color inversion result**

Figure (5): A binary image inversion result

4. Lab Experiments

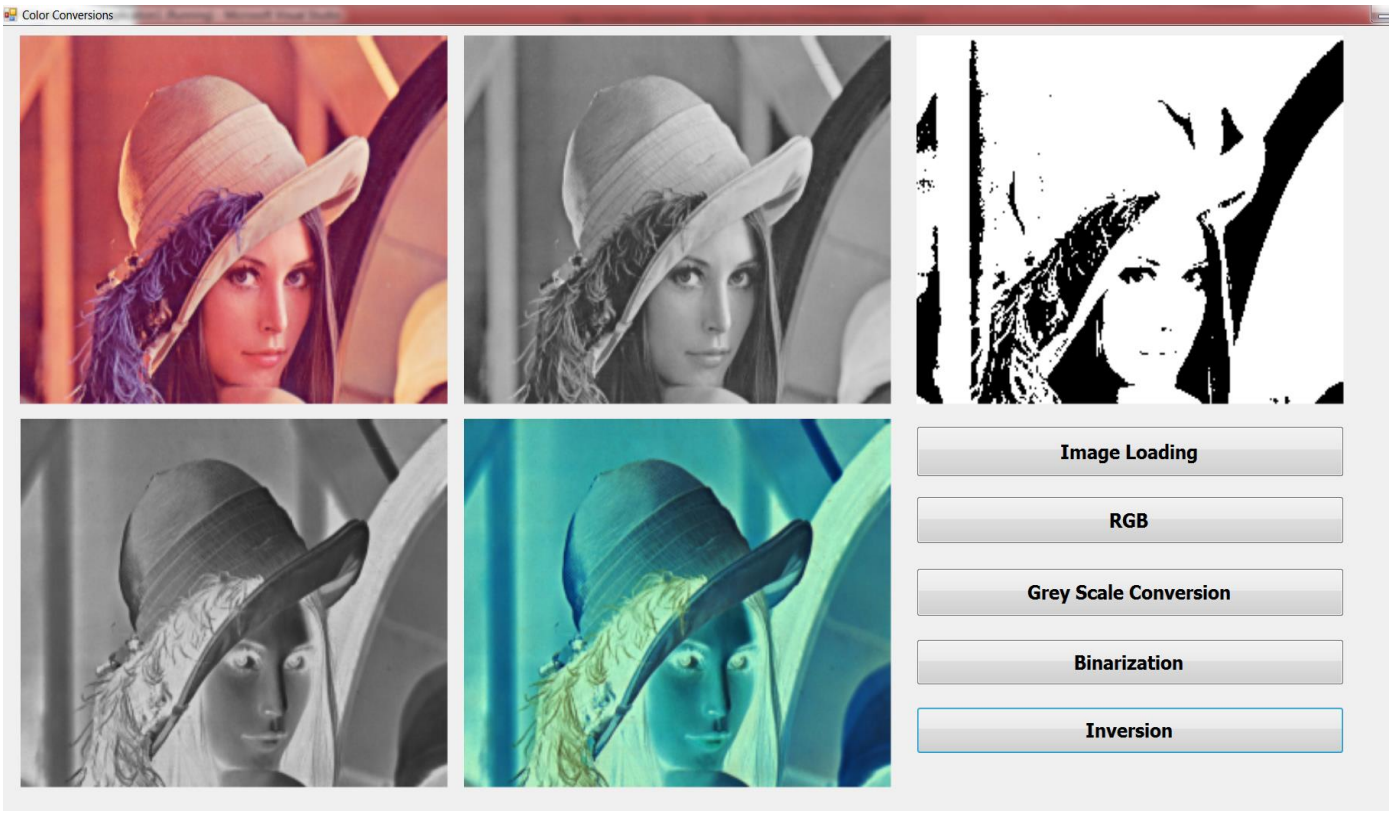
- Grey scale conversion for a true color image using:
  - Averaging method.
  - Weighted method.
- Image binarization.
- Color inversion on:
  - A true color image.
  - A grey scale image.
  - A binary image.

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 3 : Color Conversions

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3;
        int w, h;
        byte[,] red, green, blue, Grey, Binary, Inv_Grey, Inv_red, Inv_green, Inv_blue;

        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            //openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            bmp1 = new Bitmap(w, h);
            Binary = new byte[w, h];
            int th = 100;
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    if (Grey[i, j] > th)
                        Binary[i, j] = 255;
                    else
                        Binary[i, j] = 0;
                    bmp1.SetPixel(i, j, Color.FromArgb(Binary[i, j], Binary[i, j],
Binary[i, j]));
                }
            }
            pictureBox3.Image = bmp1;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;

            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];


            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);


            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));

                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;
```

```
        }

        private void button4_Click(object sender, EventArgs e)
        {
            // Average Method
            bmp1 = new Bitmap(w, h);
            Grey = new byte[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Grey[i, j] = (byte)((red[i, j] + green[i, j] + blue[i, j]) / 3);
                    bmp1.SetPixel(i, j, Color.FromArgb(Grey[i, j], Grey[i, j], Grey[i,
j]));
                }
            }
            pictureBox2.Image = bmp1;

            // Lab. Task: Weighted Method

            //bmp1 = new Bitmap(w, h);
            //Grey = new byte[w, h];
            //for (int i = 0; i < w; i++)
            //{
            //    for (int j = 0; j < h; j++)
            //    {
            //        Grey[i, j] = (byte)(0.3 * red[i, j] + 0.59 * green[i, j] + 0.11 *
blue[i, j]);
            //        bmp1.SetPixel(i, j, Color.FromArgb(Grey[i, j], Grey[i, j],
Grey[i, j]));
            //    }
            //}
            //pictureBox5.Image = bmp1;



        }

        private void button5_Click(object sender, EventArgs e)
        {
            // Grey Scale Inversion
            bmp1 = new Bitmap(w, h);
            Inv_Grey = new byte[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Inv_Grey[i, j] = (byte)(255 - Grey[i, j]);
                    bmp1.SetPixel(i, j, Color.FromArgb(Inv_Grey[i, j], Inv_Grey[i, j],
Inv_Grey[i, j]));
                }
            }
            pictureBox4.Image = bmp1;


           // True Color Inversion
            bmp1 = new Bitmap(w, h);
            Inv_red = new byte[w, h];
            Inv_green = new byte[w, h];
            Inv_blue = new byte[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
```

```
                {
                    Inv_red[i, j] = (byte)(255 - red[i, j]);
                    Inv_green[i, j] = (byte)(255 - green[i, j]);
                    Inv_blue[i, j] = (byte)(255 - blue[i, j]);
                    bmp1.SetPixel(i, j, Color.FromArgb(Inv_red[i, j], Inv_green[i, j],
Inv_blue[i, j]));
                }
            }
            pictureBox5.Image = bmp1;
        }
    }
}
```

Lab. 4: Histogram, Probability, Accumulated Probability, Mean, Standard Deviation, Histogram Moments

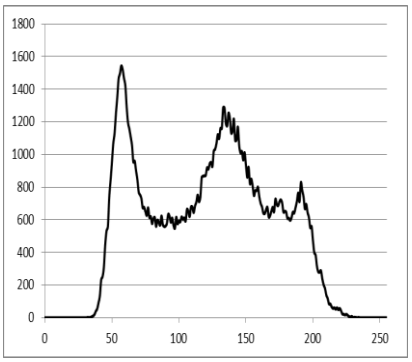
1. Image Histogram

The histogram gives an indicator to the number of pixels in an image at each different intensity value found in that image. The exact output from the operation depends upon the implementation; it may simply be a picture of the required histogram in a suitable image format, or it may be a data file of some sort representing the histogram statistics.

For an 8-bit grey scale image, there are 256 different possible intensities, so the histogram will graphically display 256 numbers showing the distribution of pixels amongst those grey scale values as shown in Figure (1).

Figure (1): Histogram for a grey scale image

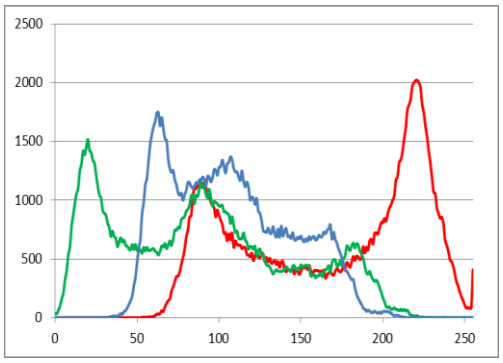
Grey scale image



Histogram chart for the grey scale image

Histogram can also be taken for color images - either by individual histogram for red, green and blue channels or a 3-D histogram with three axes representing the red, blue and green channels and brightness at each point representing the pixel count. Figure (2) shows the histogram chart for the RGB components of a true color image.

Orignal image

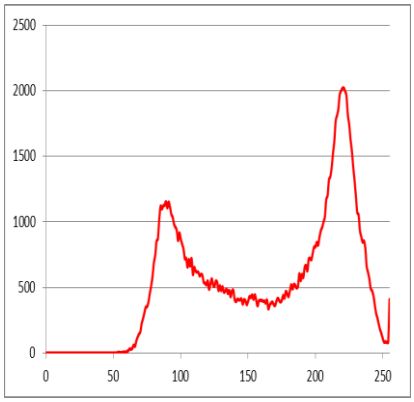


Histogram chart for the three RGB channels

Red channel

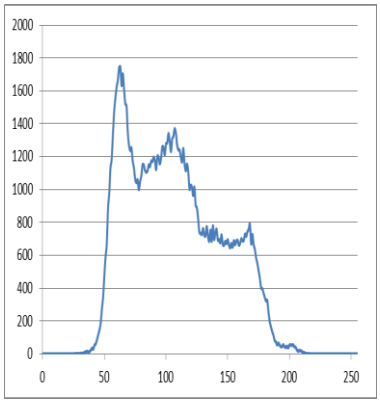
Green channel

Blue channel



Histogram chart for the red channel

Histogram chart for the green channel



Histogram chart for the blue channel

Figure (1): The histogram chart for RGB components of a true color image

2. Probability

Probability gives the count or frequency of each element in the data. For example, if we have the sample data shown in Table (1), then the probability of its data elements can be found by:

Table (1): Sample data

1	2	7	5	6
7	2	3	4	5
0	1	5	7	3
1	2	5	6	7
6	1	0	3	4

a. Computing the histogram of each data value (i.e., counting how much time this value appears in the whole data).

b. Dividing each histogram value by the total count of elements in the data.

Table (2) shows result of applying steps (**a)** and **(b**) on the data shown in Table (1).

Table (2): Probability of data values showed in Table 1

Value	**No. of Occurrence**	**Probability**
0	2	2/25 ≅ 0.08
1	4	4/25 ≅ 0.16
2	3	3/25 ≅ 0.12
3	3	3/25 ≅ 0.12
4	2	2/25 ≅ 0.08
5	4	4/25 ≅ 0.16
6	3	3/25 ≅ 0.12
7	4	4/25 ≅ 0.16
Total	25	1

3. Accumulated Probability

Accumulated probability reflects the cumulative sum of all value probabilities. Table (3) shows the accumulated probability for the data probability shown in Table (2).

Table (3): Accumulated probability for the data probability shown in Table (1)

Value	**Probability**	**Accumulated Probability**
0	0.08	0.08
1	0.16	0.24
2	0.12	0.36

3	0.12	0.48
4	0.08	0.56
5	0.16	0.72
6	0.12	0.84
7	0.16	1

4. Mean and Standard Deviation

Mean, denoted by Greek letter mu (μ), refers to the center of the data elements. It can be obtained by summing up a given set of numbers and then dividing this sum by the total number in the set as shown in the following equation:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} X_i$$

Where X_i is a data sample of N values (X_1, X_2, X_3, …,X_N).

Standard Deviation, denoted by Greek letter sigma (σ), is the measure of spread of the numbers in a set of data from its mean value. In mathematic forms, it can be given by:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \mu)^2}$$

The square value of the standard deviation is referred to *"Variance*" which is denoted by σ^2.

5. Histogram Moments

When a set of values has a sufficiently strong central tendency, that is, a tendency to cluster around some particular value, then it may be useful to characterize the set by a few numbers that are related to its *moments*, the sums of integer powers of the values. The general formula of moment with order s for data X_i containing N elements is:

The s^{th} moment = $(X_1^s + X_2^s + X_3^s + \ldots + X_N^s)/N$

The first moment (i.e., when s=1) is the mean of the data sample and the second moment (i.e., when s=2) measures how wide a distribution of the data is (the variance) while the third moment (i.e., when s=3) is skewness of data center from its median.

6. Lab Experiments

- Compute the histogram for:
  - A grey scale image
  - A true color image
- After computing the image histogram:
  - Find the probability for each element in it.
  - Find the accumulated probability.
  - Find the mean value.
  - Find the standard deviation and variance values.
  - Find the moment value (for some s order).

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 4 : Image Statistics

Design

Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3, bmp4;
        int w, h;
        byte[,] red, green, blue, Grey;
        int[] histogram_grey;
        int[] histogram_red;
        int[] histogram_green;
        int[] histogram_blue;

        public Form1()
        {
```

```
            InitializeComponent();
        }

        //Mean Calculation Function
        private void Image_Mean(int w, int h, byte[,] Image, ref double M)
        {
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    M = M + Image[i, j];
                }
            }
            M = M / (double)(w * h);
        }

        //Std Calculation Function
        private void Image_Std(int w, int h, byte[,] Image, double M, ref double S)
        {
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    S = S + ((Image[i, j] - M) * (Image[i, j] - M));
                }
            }
            S = Math.Sqrt((S / (double)(w * h)));
        }

        //Histogram Calculation Function
        public void Histogram_Cal(int wid, int hgt, byte[,] I, ref int[] Histogram)
        {
            for (int i = 0; i < wid; i++)
            {
                for (int j = 0; j < hgt; j++)
                {
                    Histogram[I[i, j]]++;
                }
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
           // openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {

            bmp1 = new Bitmap(w, h);
            Grey = new byte[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Grey[i, j] = (byte)((red[i, j] + green[i, j] + blue[i, j]) / 3);
                    bmp1.SetPixel(i, j, Color.FromArgb(Grey[i, j], Grey[i, j], Grey[i,
j]));
                }
            }
            pictureBox2.Image = bmp1;
```

```
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;
            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];


            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);
            bmp4 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                   bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));

                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;

        }



        private void button4_Click(object sender, EventArgs e)
        {

            double  GreyMean = 0;
            Image_Mean(w, h, Grey, ref GreyMean);

            double  GreyStd = 0;
            Image_Std(w, h, Grey, GreyMean, ref GreyStd);

            label1.Text ="Grey Scale Image Mean = " + GreyMean;
            label2.Text =  "Grey Scale Image Std = " + GreyStd;

        }

        private void button5_Click(object sender, EventArgs e)
        {
            // Grey Scale Histogram
            histogram_grey = new int[256];

            Histogram_Cal(w, h, Grey, ref histogram_grey);

            int max = 0;
```

```
            for (int i = 0; i < 256; i++)
            {
                if (max < histogram_grey[i])
                    max = histogram_grey[i];
            }

            int histHeight = 128;
            Bitmap img = new Bitmap(256, histHeight + 10);
            Graphics g = Graphics.FromImage(img);

            for (int i = 0; i < 256; i++)
            {
                float pct = histogram_grey[i] / (float)max;   // What percentage of the
max is this value?
                Point p1 = new Point(i, img.Height - 5);
                Point p2 = new Point(i, img.Height - 5 - (int)(pct * histHeight));
                g.DrawLine(Pens.Black, p1, p2);// Use that percentage of the height

            }

            pictureBox3.Image = img;


            // Task: RGB Channels Histogram
            histogram_red = new int[256];
            histogram_green = new int[256];
            histogram_blue = new int[256];

            Histogram_Cal(w, h, red, ref histogram_red);

            Histogram_Cal(w, h, red, ref histogram_red);
            Histogram_Cal(w, h, green, ref histogram_green);
            Histogram_Cal(w, h, blue, ref histogram_blue);

            int maxr = 0, maxg = 0, maxb = 0;
            for (int i = 0; i < 256; i++)
            {
                if (maxr < histogram_red[i])
                    maxr = histogram_red[i];
                if (maxg < histogram_green[i])
                    maxg = histogram_green[i];
                if (maxb < histogram_blue[i])
                    maxb = histogram_blue[i];
            }

           img = new Bitmap(256, histHeight + 10);
            g = Graphics.FromImage(img);

            for (int i = 0; i < 256; i++)
            {
                float pct = histogram_red[i] / (float)maxr;   // What percentage of the
max is this value?
                g.DrawLine(Pens.Red,
                    new Point(i, img.Height - 5),
                    new Point(i, img.Height - 5 - (int)(pct * histHeight))  // Use that
percentage of the height
                    );

                pct = histogram_green[i] / (float)maxg;   // What percentage of the max
is this value?
                g.DrawLine(Pens.Green,
                    new Point(i, img.Height - 5),
                    new Point(i, img.Height - 5 - (int)(pct * histHeight))  // Use that
percentage of the height
```

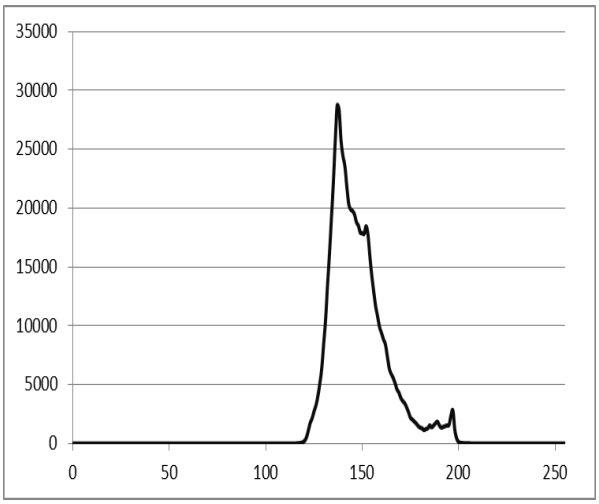
```
                    );

                pct = histogram_blue[i] / (float)maxb;   // What percentage of the max
is this value?
                g.DrawLine(Pens.Blue,
                    new Point(i, img.Height - 5),
                    new Point(i, img.Height - 5 - (int)(pct * histHeight))  // Use that
percentage of the height
                    );
            }

            pictureBox4.Image = img;


        }
    }
}
```

Lab. 5: Linear Contrast Stretching

1. Contrast Term Meaning

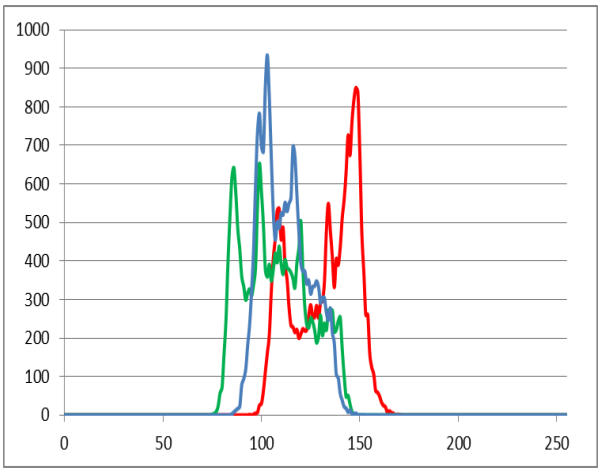
The term *'contrast'* refers to the amount of color or grey scale differentiation that exists between various image features. Images having a higher contrast level generally display a greater degree of color or grey scale variation than those of lower contrast. Figure (1) shows examples for low-contrast images (for a grey scale image and true color image), along with their histogram.

(a) Low contrast grey scale image



Histogram chart for (a)

(b) Low contrast true color image



Histogram chart for (b)

Figure (1): Low contrast images with their histogram

2. Linear Contrast Stretching

Contrast stretching (also called Normalization) attempts to improve an image by stretching the range of intensity values it contains to make full use of possible values. The following function is used to stretch the contrast of the image:

$$\mathbf{I}_s = \begin{cases} G_{min} & if\ I < G_{min} \\ 255\left(\dfrac{\mathbf{I} - \mathbf{G_{min}}}{\mathbf{G_{max}} - \mathbf{G_{min}}}\right) & if\ G_{min} \le I \le G_{max} \\ G_{max} & if\ \ I > G_{max} \end{cases}$$

Where, G_{min} & G_{max} is the minimum and maximum intensity values in the original image, respectively.

There are different methods to assess the values G_{min} & G_{max} , which in turn affects the outcomes of linear stretching method.

(i) *Minimum-Maximum Linear Contrast Stretching*

In min-max linear contrast stretching, the histogram of the original image is examined to determine the values of G_{min} & G_{max} which represent minimum and maximum values of the data. Then, the contrast stretching is applied by mapping the minimum gray level G_{min} in the image to zero and the maximum gray level G_{max} to 255, the other gray levels are remapped linearly between 0 and 255. Figure (2) shows result of applying min-max contrast stretching method on low-contrast images that shown in Figure (1).

(ii) *Percentage Linear* Contrast Stretching

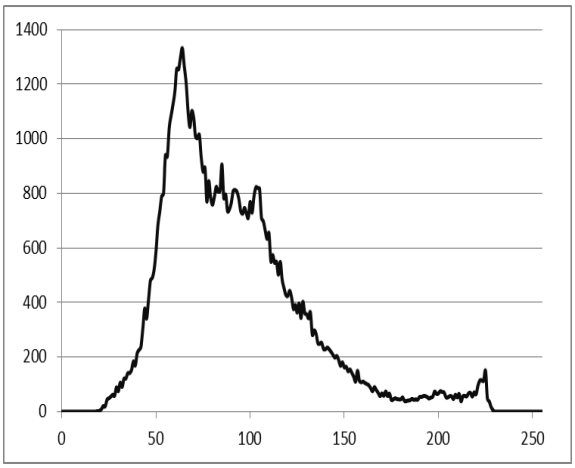
It is similar to the minimum-maximum linear contrast stretch except this method uses specified minimum and maximum values that lie in a certain percentage of pixels from the mean of the histogram. The values of G_{min} and G_{max} are determined using the following equations:

$$G_{min} = \mu - \alpha\,\sigma\,, \qquad G_{max} = \mu + \alpha\,\sigma$$

Where μ, σ are the mean and standard deviation values, respectively, of the image. The parameter α used to control the strength of achieved linear

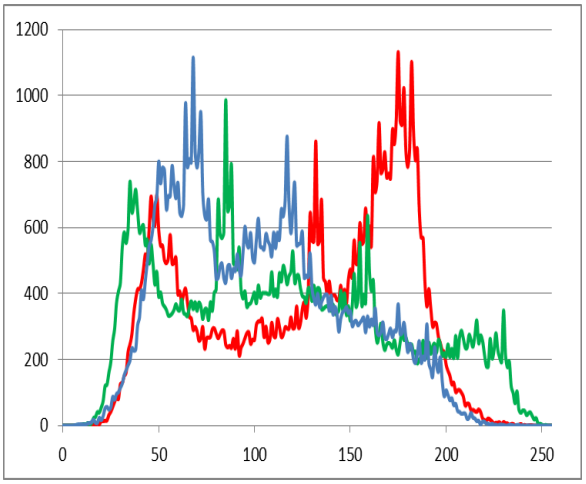
extent. Figure (3) shows result of applying percentage contrast stretching method on the low-contrast images that shown in Figure (1) with two different α values (i.e., 0.5 and 1.5).

(a) Min-max contrast stretching result on the image shown in Figure 1a



Histogram chart for (a)

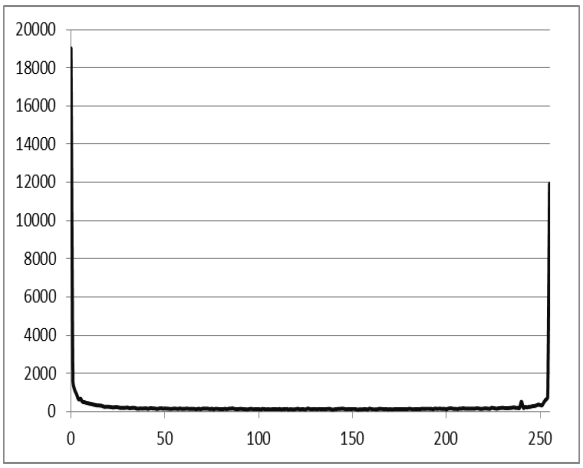
(b) Min-max contrast stretching result on the image shown in Figure 1b



Histogram chart for (b)

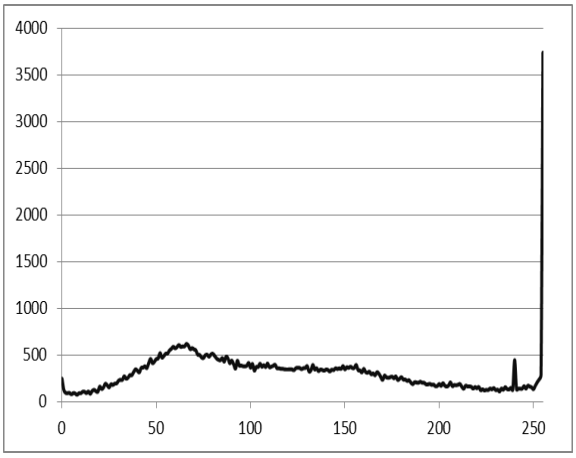
Figure (2): Result of applying min-max contrast stretching method on the low-contrast images that shown in Figure (1)

(a-1) Percentage contrast stretching result on the image shown in Figure 1 (a) with α=0.5



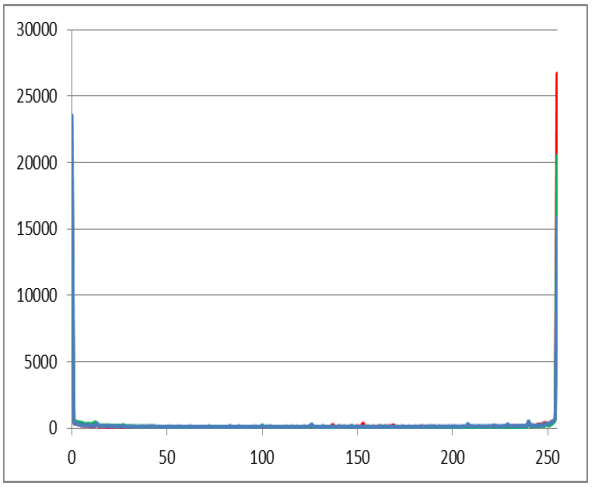
Histogram chart for (a-1)

(a-2) Percentage contrast stretching result on the image shown in Figure 1a with α=1.5



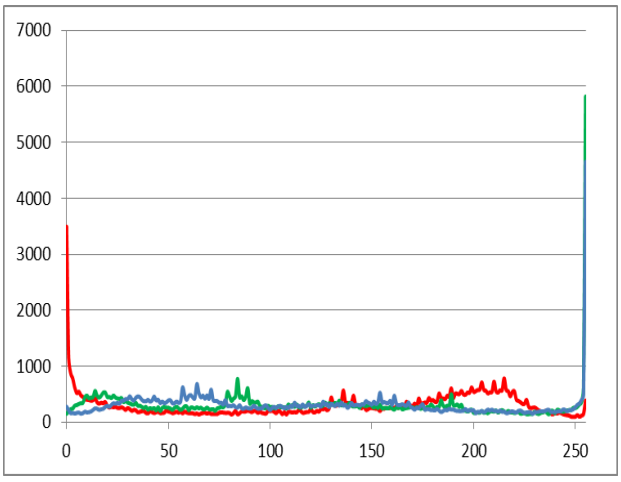
Histogram chart for (a-2)

(b-1) Percentage contrast stretching result on the image shown in Figure 1b with α=0.5



Histogram chart for (b-1)

(b-2) Percentage contrast stretching result on the image shown in Figure 1b with α=1.5



Histogram chart for (b-2)

Figure (3): Result of applying percentage contrast stretching method for the low-contrast images that shown in Figure (1)

3. Lab Experiments

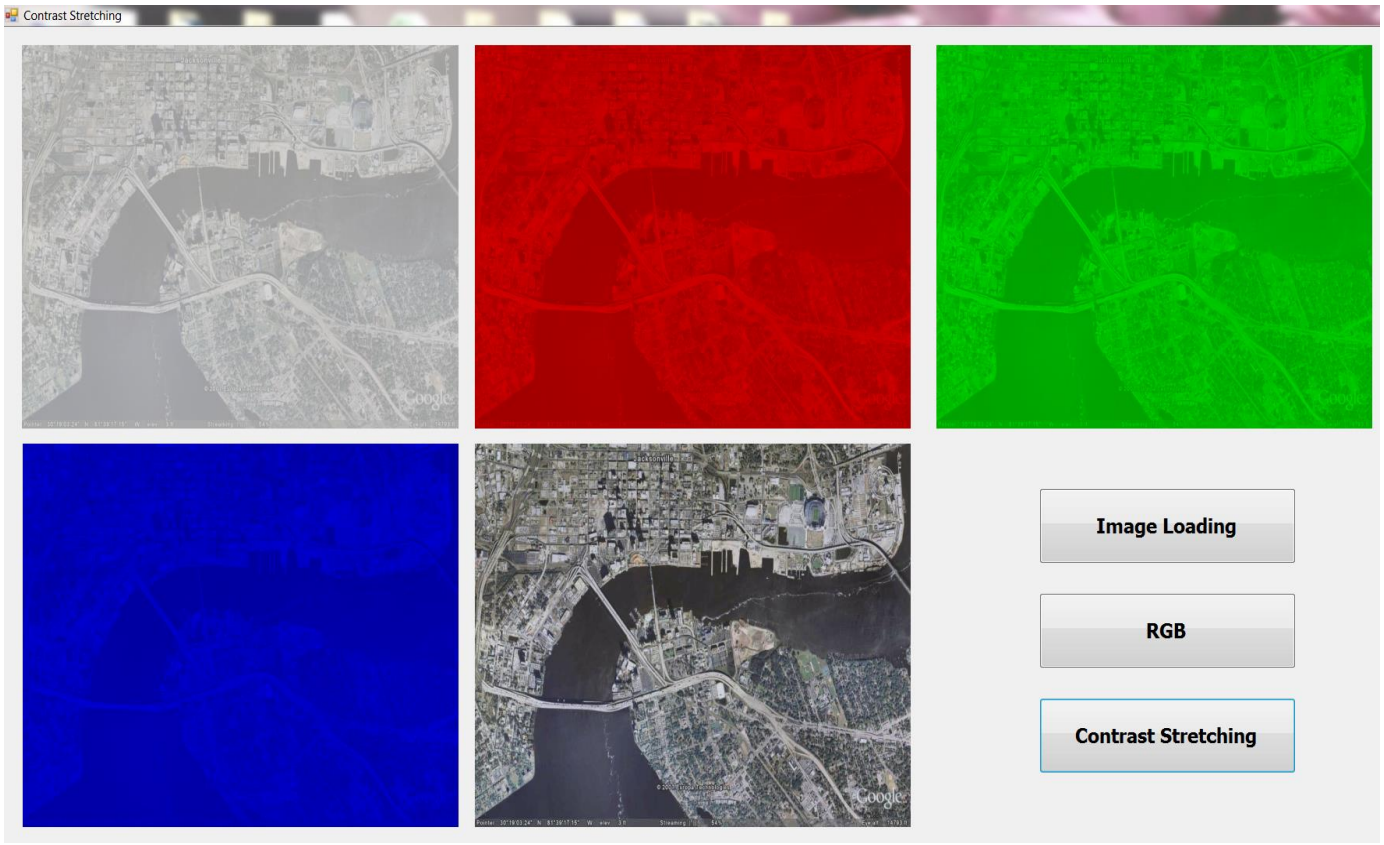
- Try to apply linear contrast stretching using:
  - Min-max contrast stretching method.
  - Percentage contrast stretching method.

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 5 : Contrast Stretching

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3;

        int w, h;

        byte[,] red;
        byte[,] green;
        byte[,] blue;
```

```
        byte[,] MMContrast_red;
        byte[,] MMContrast_green;
        byte[,] MMContrast_blue;

        public Form1()
        {
            InitializeComponent();
        }

        //Min-Max Contrast Stretching Function
        private void MinMaxContrastStreching(int wid, int hgt, byte[,] Image, ref
byte[,] SI)
        {
            byte Min = Image[0, 0];
            byte Max = Image[0, 0];
            for (int i = 0; i < wid; i++)
            {
                for (int j = 0; j < hgt; j++)
                {
                    if (Max < Image[i, j])
                        Max = Image[i, j];
                    if (Min > Image[i, j])
                        Min = Image[i, j];
                }
            }
            byte[] lookup = new byte[256];
            for (int i = 0; i < 256; i++)
            {
                lookup[i] = (byte)(255.0 * ((i - Min) / (float)(Max - Min)));
            }
            for (int i = 0; i < wid; i++)
            {
                for (int j = 0; j < hgt; j++)
                {
                    SI[i, j] = lookup[Image[i, j]];
                }
            }
        }


        private void button1_Click(object sender, EventArgs e)
        {
            //openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            MMContrast_red = new byte[w, h];
            MMContrast_green = new byte[w, h];
            MMContrast_blue = new byte[w, h];

            MinMaxContrastStreching(w, h, red, ref  MMContrast_red);
            MinMaxContrastStreching(w, h, green, ref  MMContrast_green);
            MinMaxContrastStreching(w, h, blue, ref  MMContrast_blue);

            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
                for (int j = 0; j < h; j++)
                    bmp2.SetPixel(i, j, Color.FromArgb(MMContrast_red[i, j],
MMContrast_green[i, j], MMContrast_blue[i, j]));
```

```
            pictureBox5.Image = bmp2;


        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;

            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];

            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));

                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;

        }
    }
}
```

Baghdad University/ College of Science
Department of Computer Science

Image Processing Lab
Third Class

Lab. 6: Histogram Equalization

1. Histogram Equalization

Histogram equalization is a technique for adjusting image intensities to enhance its contrast. It Increases local contrast by spreading out the intensity histogram, so that the new histogram is wider and more uniform in term of distribution of intensity values than original image histogram. Histogram equalization can be done in three steps:

a. Find the probability of each image intensity value.

b. Find the accumulated probability using probability values resulted from (a).

c. Remap pixels by multiplying accumulated probability with the maximum value of the new scale.

Example:

Consider a sub image has the following values:

4	1	3	2
3	1	1	1
0	1	7	2
1	1	2	2

The table shown below illustrates the steps of applying histogram equalization on the above sub image using the same old scale (old scale=3 bits; i.e., the max value =7):

Intensity	Histogram	Probability	Accumulated Probability	New Intensity
0	1	1/16=0.0625	0.0625	0.0625*7=0.4375≅**0**
1	7	7/16=0.4375	0.5	0.5*7=3.5≅**4**
2	4	4/16=0.25	0.75	0.75*7=5.25≅**5**
3	2	2/16=0.125	0.875	0.875*7=6.125≅**6**

4	1	1/16=0.0625	0.9375	0.9375*7=6.5625≅**7**
7	1	1/16=0.0625	1	1*7=7≅ **7**

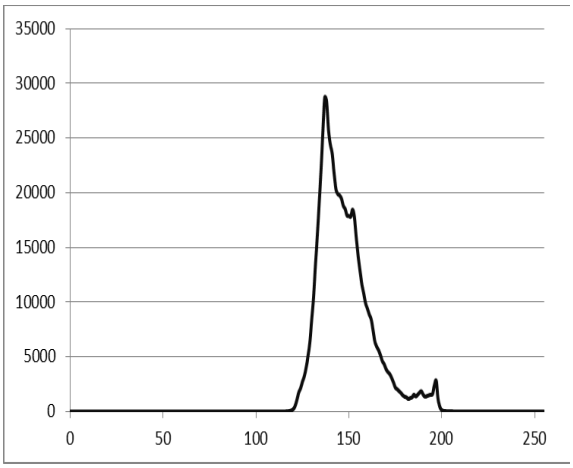
And the new sub image will be:

7	**4**	**6**	**5**
6	**4**	**4**	**4**
0	**4**	**7**	**5**
4	**4**	**5**	**5**

Table (1) shows the results of applying histogram equalization on a grey scale image and a true color image with different new scale values.

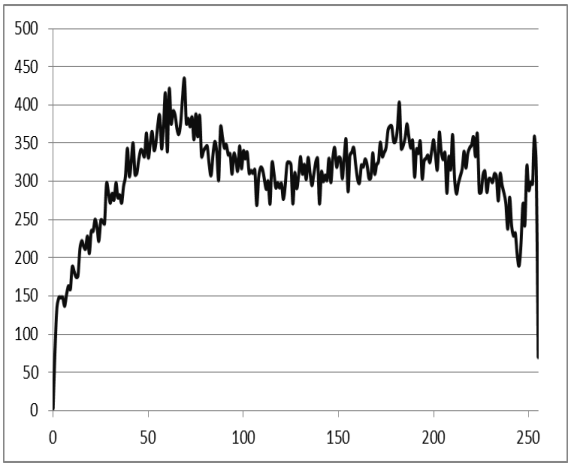
Table (1): The result of applying histogram equalization on a grey scale image and a true color image with different new scale values

(a) Original grey scale image



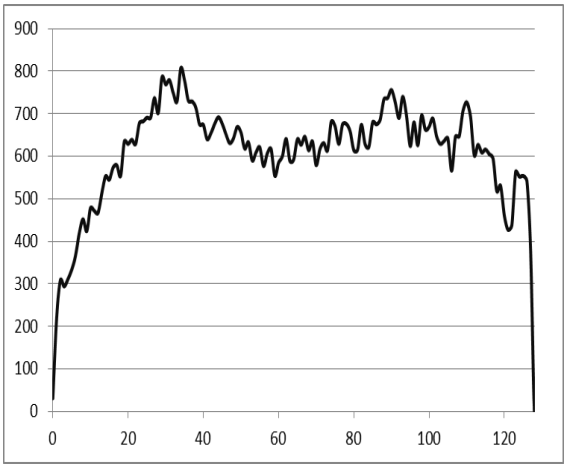
Histogram chart for (a)

(a-1) Histogram equalization result with new grey scale=8 bits (i.e., the max value=255)



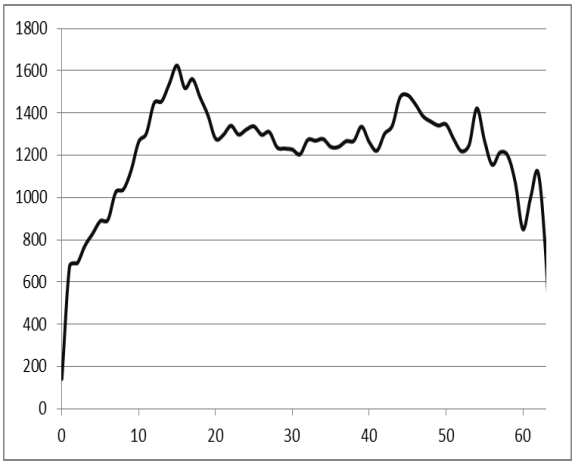
Histogram chart for (a-1)

(a-2) Histogram equalization result with new grey scale=7 bits (i.e., the max value=127)



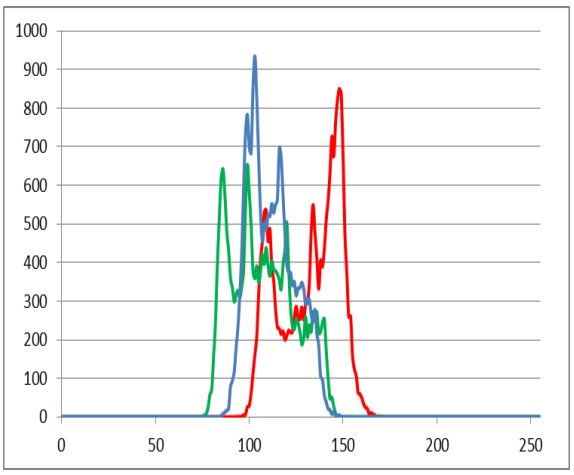
Histogram chart for (a-2)

(a-3) Histogram equalization on (a) result with new grey scale=6 bits (i.e., the max value=63)



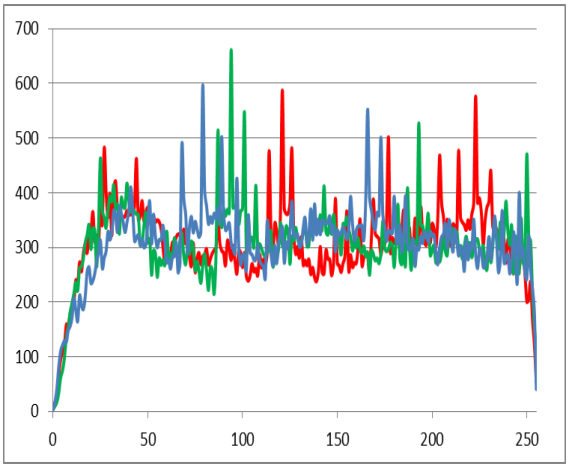
Histogram chart for (a-3)

(b) Original true color image



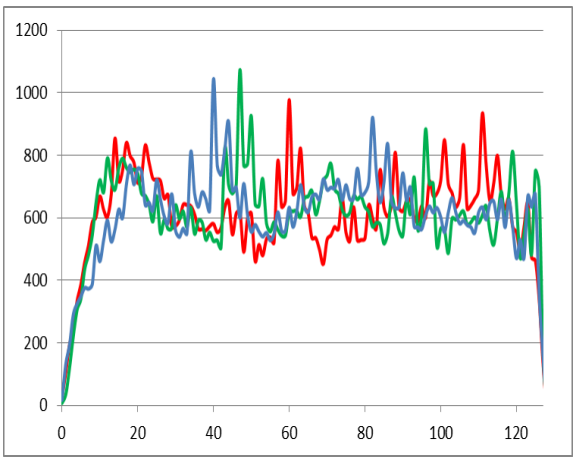
Histogram chart for (b)

(b-1) Histogram equalization result with new scale=8 bits for each color band (i.e., the max value=255)



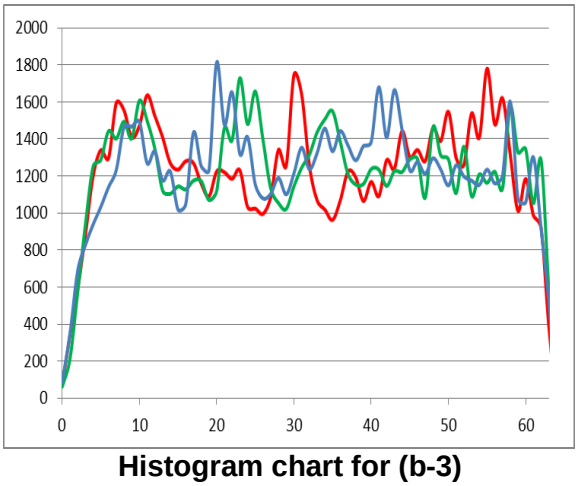
Histogram chart for (b-1)

(b-2) Histogram equalization result with new scale=7 bits for each color band (i.e., the max value=163)



Histogram chart for (b-2)

(b-3) Histogram equalization result with new scale=6 bits for each color band (i.e., the max value=63)



Histogram chart for (b-3)

2. Lab. Experiments

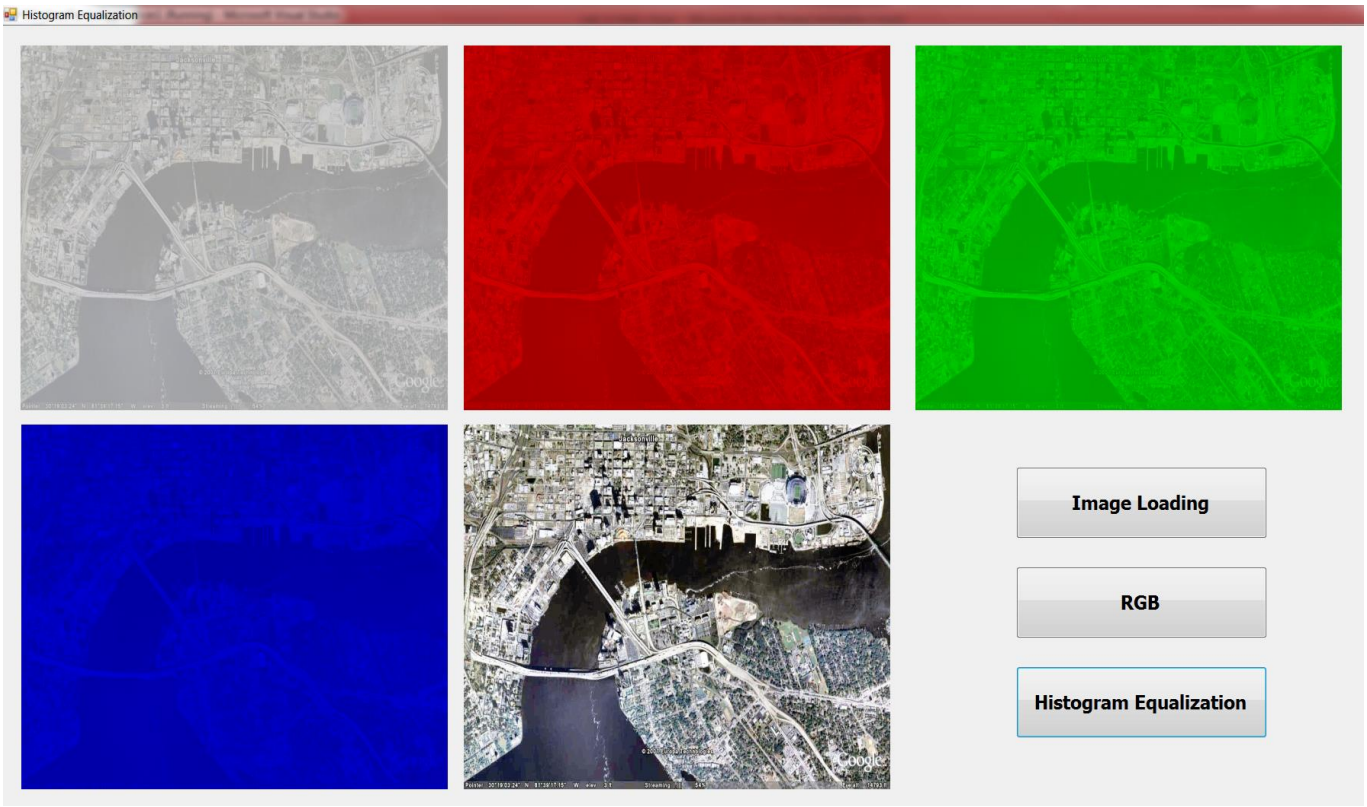
- Histogram Equalization on:
  - A grey scale image.
  - A true color image.

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 6 : Histogram Equalization

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3;
        int w, h;
        byte[,] red;
        byte[,] green;
        byte[,] blue;
        byte[,] histogramEquliz_red;
        byte[,] histogramEquliz_green;
        byte[,] histogramEquliz_blue;
```

```
        public Form1()
        {
            InitializeComponent();
        }

        public void Histogram_Cal(int wid, int hgt, byte[,] I, ref int[] Histogram)
        {
            for (int i = 0; i < wid; i++)
            {
                for (int j = 0; j < hgt; j++)
                {
                    Histogram[I[i, j]]++;
                }
            }
        }

        public void Histogram_Equalization(int wid, int hgt, byte[,] Image, ref byte[,]
EI, int OPI, int NPI)
        {
            //Compute  histogram of the image
            int OIR = (int)Math.Pow(2, OPI);
            int[] His = new int[OIR];
            Histogram_Cal(wid, hgt, Image, ref His);

            //Compute probablity  of the image
            double[] Prob = new double[OIR];
            for (int i = 0; i < 256; i++)
            {
                Prob[i] = His[i] / (double)(w * h);
            }

            //  Compute Acmulative probablity  of the image
            double[] AcmP = new double[OIR];
            AcmP[0] = Prob[0];

            for (int i = 1; i < 256; i++)
            {
                AcmP[i] = AcmP[i - 1] + Prob[i];
            }

            //Remapping pixels using histogram equalization
            int NIR = (int)Math.Pow(2, NPI);
            for (int i = 0; i < wid; i++)
            {
                for (int j = 0; j < hgt; j++)
                {
                    EI[i, j] = (byte)Math.Round((NIR - 1) * AcmP[Image[i, j]], 0);

                }
            }
        }



        private void button1_Click(object sender, EventArgs e)
        {
            //openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
```

```
        {
            histogramEquliz_red = new byte[w, h];
            histogramEquliz_green = new byte[w, h];
            histogramEquliz_blue = new byte[w, h];

            Histogram_Equalization(w, h, red, ref histogramEquliz_red, 8, 8);
            Histogram_Equalization(w, h, green, ref histogramEquliz_green, 8, 8);
            Histogram_Equalization(w, h, blue, ref histogramEquliz_blue, 8, 8);

            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
                for (int j = 0; j < h; j++)
                    bmp2.SetPixel(i, j, Color.FromArgb(histogramEquliz_red[i, j],
histogramEquliz_green[i, j], histogramEquliz_blue[i, j]));
            pictureBox5.Image = bmp2;


        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;

            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];

            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));


                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;

        }
    }
}
```

Lab. 7: Gamma Correction

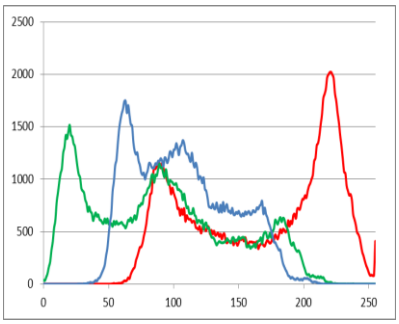
1. Luminance

Luminance is the property that describes the brightness of the light in the image. Figure (1) explains luminance meaning with different images have different brightness (for a grey scale image and a true color image), along with their histogram.

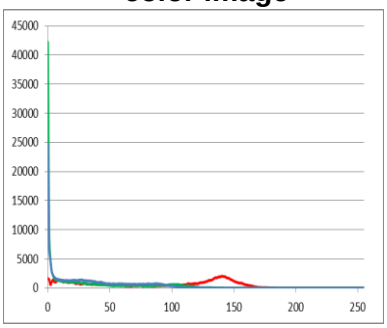
(a1) A true color Image

(a2) Low brightness true color image

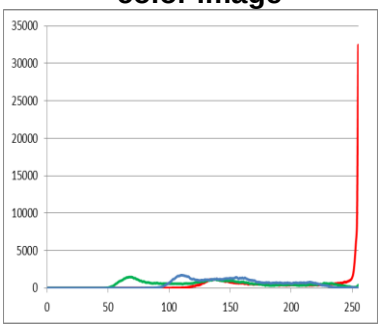
(a2) High brightness true color image



Histogram chart for (a-1)



Histogram chart for (a-2)

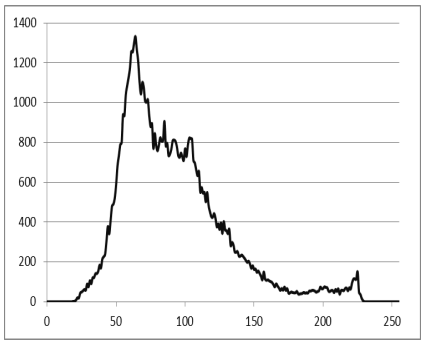


Histogram chart for (a-3)

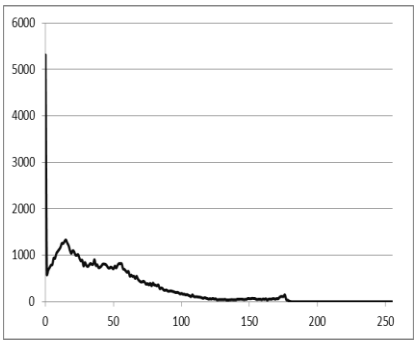
(b-1) A grey scale Image

(b-2) Low brightness grey scale image

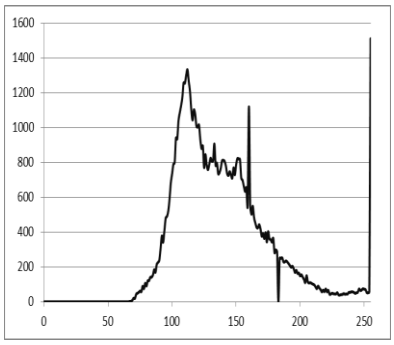
(b-3) High brightness grey scale image



Histogram chart for (b1)



Histogram chart for (b2)



Histogram chart for (b3)

Figure (1): Low contrast images with their histogram

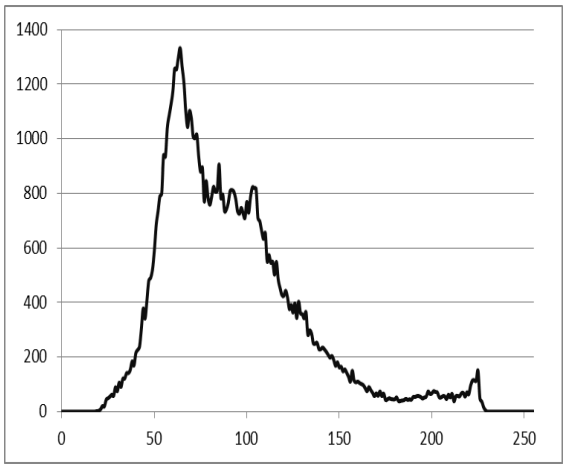
2. Gamma Mapping

In many cases, different camera or video recorder devices do not correctly capture luminance or different display devices (monitor, phone screen, TV) do not display luminance correctly neither. Therefore, there is a need to correct luminance conditions. *Gamma correction function* is used to correct image's luminance. It is usual to add a prime to signals that represent gamma correction by rising the signals to the power called gamma (γ) before transmission. For image intensity, I, the general equation to achieve gamma correction (I_G) value is:

$$\boldsymbol{I_G(x,y) = 255\left(\frac{I(x,y)}{255}\right)^{1/\gamma}}$$

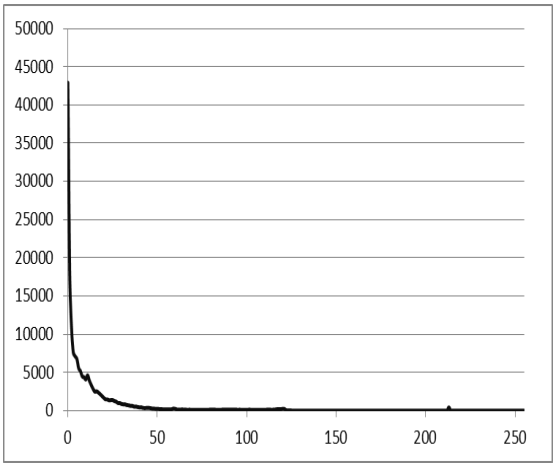
For a grey scale image, gamma correction function is applied on each pixel intensity. If the image is of a true color then gamma correction function is applied for each band of RGB colors. Figure (2) shows results of applying gamma correction on a grey scale image with different gamma values.

(a) Original image



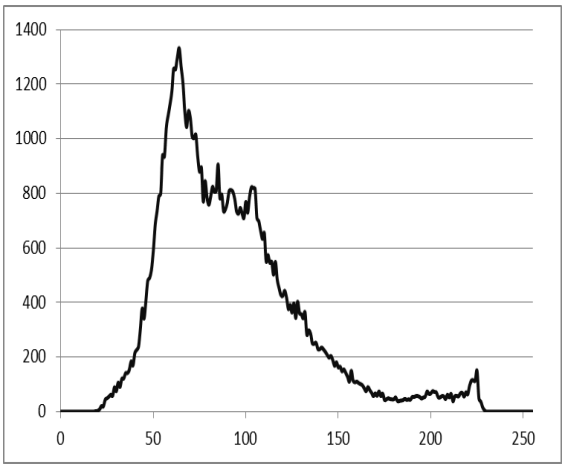
Histogram chart for (a)

(b) Gamma correction result with γ=0.5



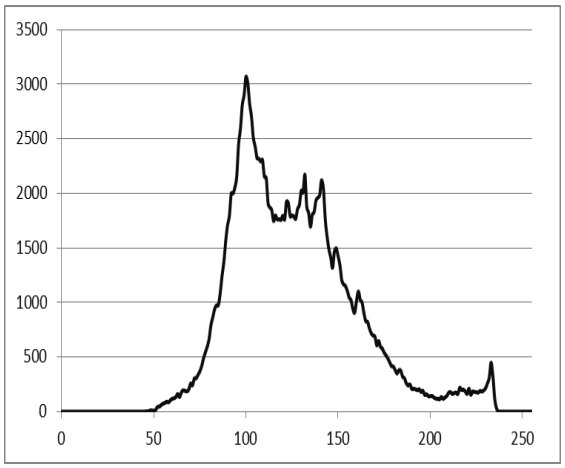
Histogram chart for (b)

(c) Gamma correction result with γ=1



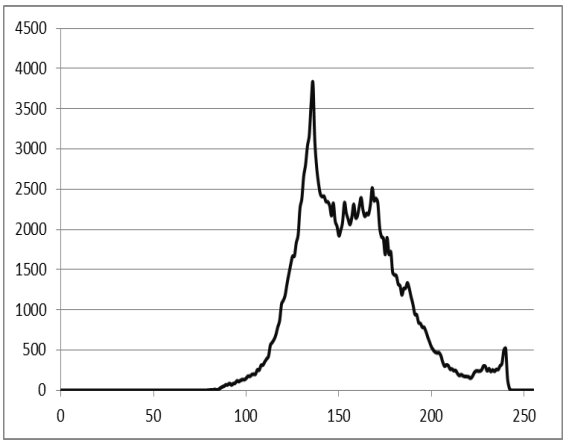
Histogram chart for (c)

(d) Gamma correction result with γ=1.5



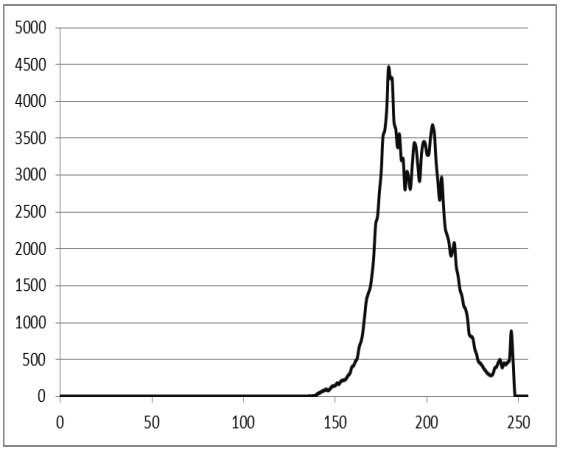
Histogram chart for (d)

(e) Gamma correction result with γ=2.2



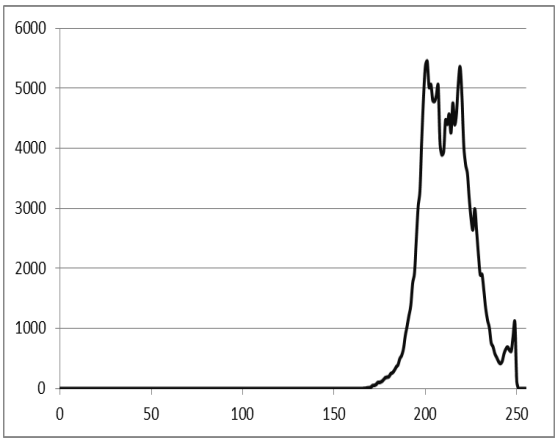
Histogram chart for (e)

(f) Gamma correction result with γ=4



Histogram chart for (f)

(g) Gamma correction result with γ=6



Histogram chart for (g)

Figure (2): Results of applying gamma correction on a grey scale image

3. Lab. Experiments

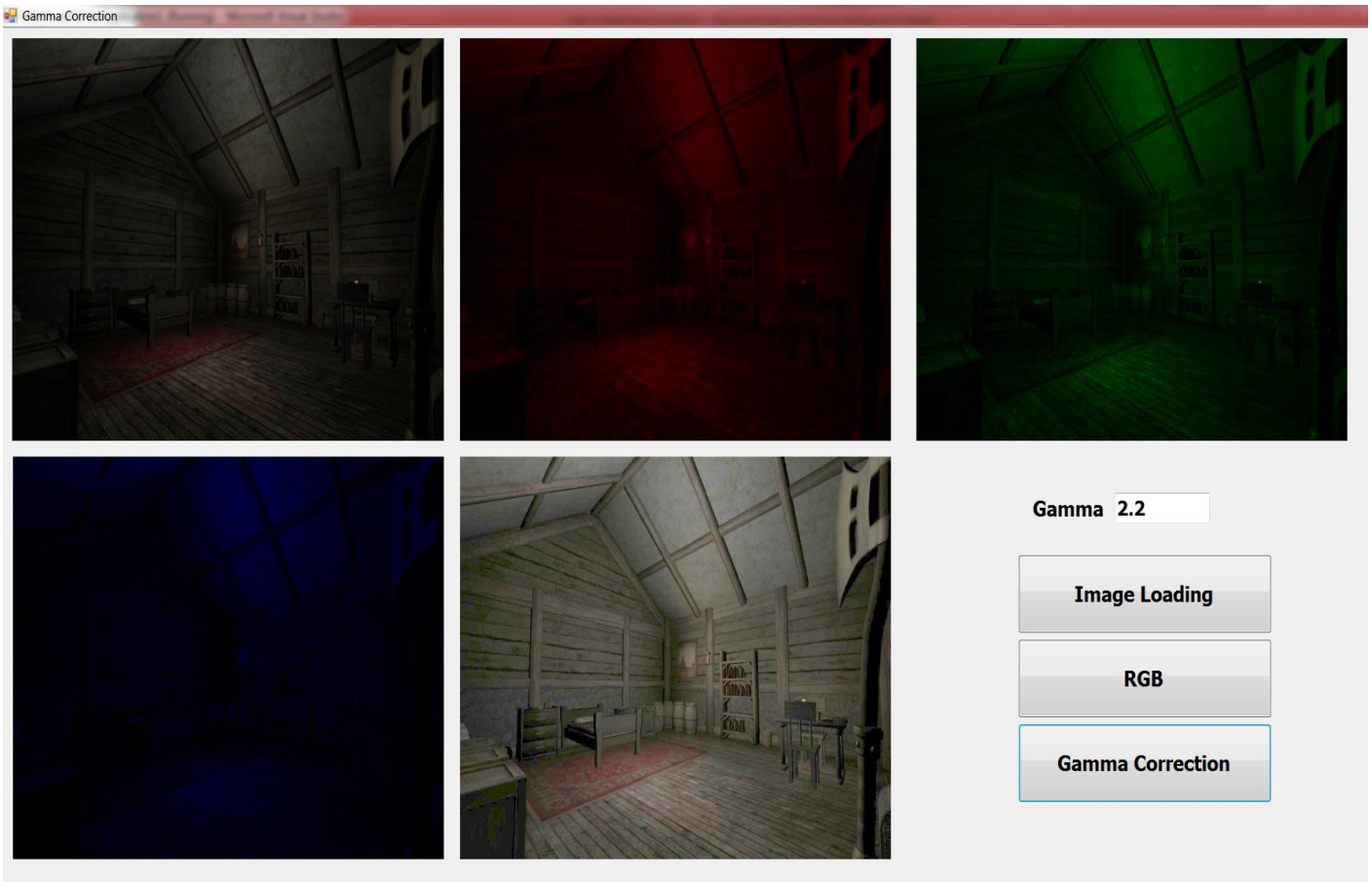
- Try to apply Gamma correction on:
  - A grey scale image.
  - A true color image.

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 7 : Gamma Correction

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3;
        int w, h;
        byte[,] red;
        byte[,] green;
        byte[,] blue;
```

```
        byte[,] GammaCorr_red;
        byte[,] GammaCorr_green;
        byte[,] GammaCorr_blue;

        public Form1()
        {
            InitializeComponent();
        }

        //Gamma Correction Function
        public void Gamma_Correction(int wid, int hgt, byte[,] I, ref byte[,] CI,
double Gamma)
        {
            byte[] lookup = new byte[256];
            for (int i = 0; i < 256; i++)
            {
                lookup[i] = (byte)(Math.Pow((i / 255f), (1 / Gamma)) * 255);
            }
            for (int i = 0; i < wid; i++)
            {
                for (int j = 0; j < hgt; j++)
                {
                    CI[i, j] = lookup[I[i, j]];
                }
            }
        }


        private void button1_Click(object sender, EventArgs e)
        {
           // openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            GammaCorr_red = new byte[w, h];
            GammaCorr_green = new byte[w, h];
            GammaCorr_blue = new byte[w, h];

            double m = Convert.ToDouble(textBox1.Text);

            Gamma_Correction(w, h, red, ref  GammaCorr_red, m);
            Gamma_Correction(w, h, green, ref  GammaCorr_green, m);
            Gamma_Correction(w, h, blue, ref  GammaCorr_blue, m);

            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
                for (int j = 0; j < h; j++)
                    bmp2.SetPixel(i, j, Color.FromArgb(GammaCorr_red[i, j],
GammaCorr_green[i, j], GammaCorr_blue[i, j]));

            pictureBox5.Image = bmp2;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;
            red = new byte[w, h];
            green = new byte[w, h];
```

```
            blue = new byte[w, h];


            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));

                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;

        }
    }
}
```

Lab. 8 & 9: Linear Smoothing Filters (Mean, Weighted Mean)

1. Image Noise Definition

Noise represents unwanted information, which degrades the image quality. During image acquisition or transmission, several factors are responsible for introducing noise in the image. Depending on the type of disturbance, the noise can affect the image to different extent. Image noise can be classified as:

A. Impulse Noise (Salt and Pepper Noise)

Impulse noise is a special type of noise which can have many different origins. Images are often corrupted by impulse noise caused by transmission errors, faulty memory locations or timing errors in analog-to-digital conversion. Salt and pepper noise is one type of impulse noise which can corrupt the image, where the noisy pixels can take only the maximum and minimum values in the dynamic range. Figures (1) and (2) show an example of images affected by impulse noise for grey scale and true color, respectively.

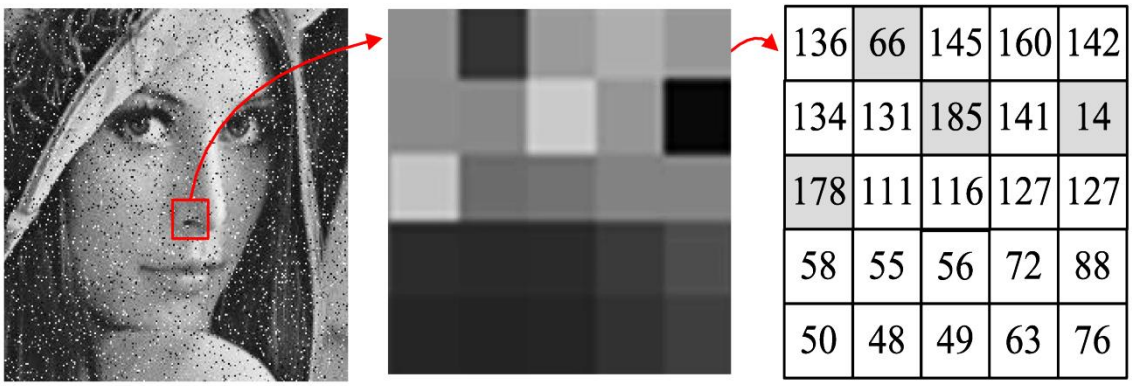


Figure (1): An example of grey scale image affected by impulse noise

Figure (2): An example of true color image affected by impulse noise

B. Gaussian Noise (Amplifier Noise)

This noise model is additive in nature and follow Gaussian distribution. Meaning that each pixel in the noisy image is the sum of the true pixel value and a random, Gaussian distributed noise value. The noise is independent of intensity of pixel value at each point. Figure (3) shows an example of grey scale image affected by Gaussian noise.

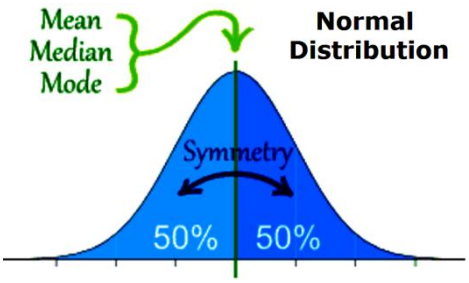


Figure (2): An example of grey scale image affected by Gaussian noise

C. Speckle Noise

This noise can be modeled by random value multiplications with pixel values of the image; it can be expressed as:

$$J = (1+ n) \times I$$

Where, J is the speckle noisy image distribution, I is the input image and n is a uniform noise. Figure (4) shows an example of grey scale image affected by speckle noise.

Figure (4): An example of grey scale image affected by speckle noise

2. Image Mask (Filter) Meaning

Image mask (Filter) is a 2D matrix that is slid across the input image where the sum of products is taken for every pixel of the image and mask matrix (this operation is called "***Convolution***"). Each product is the color value of the current pixel or a neighbor of it, with the corresponding value of the filter matrix. The center of the filter matrix has to be multiplied with the current pixel, the other elements of the filter matrix with corresponding neighbor pixels. The size of the filter must be odd, so that it has a center, for example 3x3, 5x5, 7x7, etc. Figure (5) illustrates filter meaning using filter of size 3×3 mask.

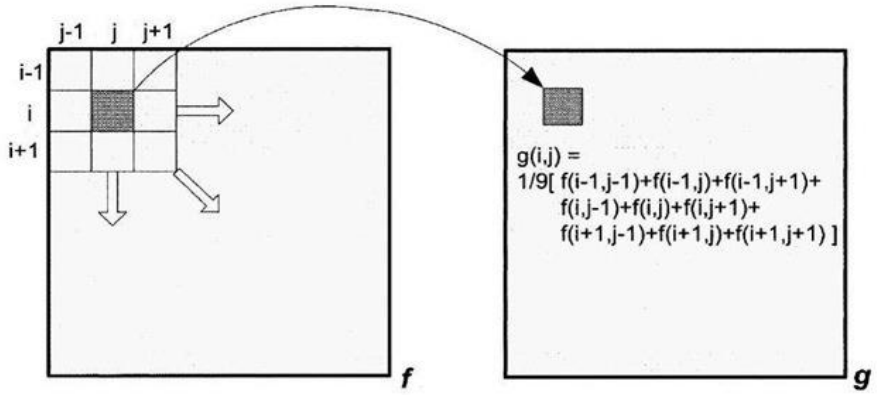


Figure (5): Filter meaning using filter of size 3×3 mask

3. Image De- Noising

Image de-noising plays a vital role in digital image processing. There are many schemes for removing noise from images. The good de-noising scheme must able to retrieve as much of image details even though the image is highly affected by noise. In common, there are two types of image de-noising filters, linear filters and nonlinear filters.

4. Image De-nosing using Linear Smoothing Filters

Smoothing filters are simple and fast filters, which used for image de-noising. They are explicitly designed to remove *impulse noise* (*i.e.*, isolated pixels have low or high pixel intensity (*e.g.*, salt and pepper noise)) and, therefore, they are less effective at removing *additive noise* (*e.g.* Gaussian noise) from an image.

The main benefit of using linear noise removing filters is the speed while the limitation of the linear filters is the filters are not able to preserve edges of the images in an efficient manner.

However, there are two main smoothing filters: mean filter and weighted mean filter.

A. Mean Filter

Mean filter, or *average filter* is windowed filter of linear class that is used to smooth the image. Mean filter intends to use the mean (or average) value of pixel neighbors, including itself as a replacement of the original pixel in an input image. Like other convolution filters, mean filter is based around a kernel, which represents the size and shape of the neighborhood to be sampled in calculation of mean value. The 3x3 averaging filter is:

$$f_{3\times3} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Figure (6) gives a numerical example that demonstrates the result of applying mean filter on a sub-image.

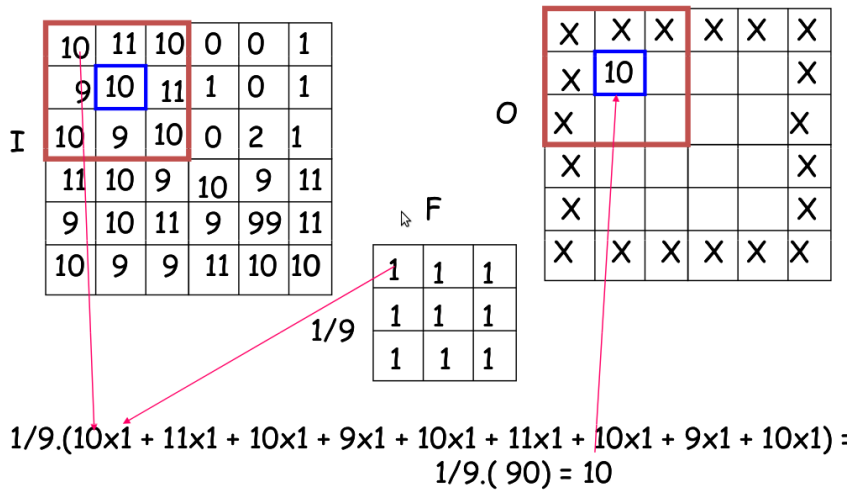


Figure (6): A numerical example for applying mean filter on a sub-image

Tables (1) and (2) show results of applying mean filter on noisy images (for the three previously mentioned types of the noise) with filter size 3x3 and 5x5, respectively.

Table (1): Results of applying mean filter on noisy images with filter size 3x3

Noise type	Noisy image	Mean filter of size 3x3 for 1 times	Mean filter of size 3x3 for 2 times
Salt& Pepper Noise			
Gaussian Noise			

Speckle Noise			

Table (2): Results of applying mean filter on noisy images with filter size 5x5

Noise type	Noisy image	Mean filter of size 5x5 for 1 times	Mean filter of size 5x5 for 2 times
Salt& Pepper Noise			
Gaussian Noise			
Speckle Noise			

B. Weighted Mean Filter

Instead of averaging all the pixel values in the window, weighted mean filter gives the closer-by pixels higher weighting, and far-away pixels lower weighting as following:

$$f_{3\times3} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Figure (7) gives a numerical example that shows the result of applying mean filter and weighted mean on a sub-image.

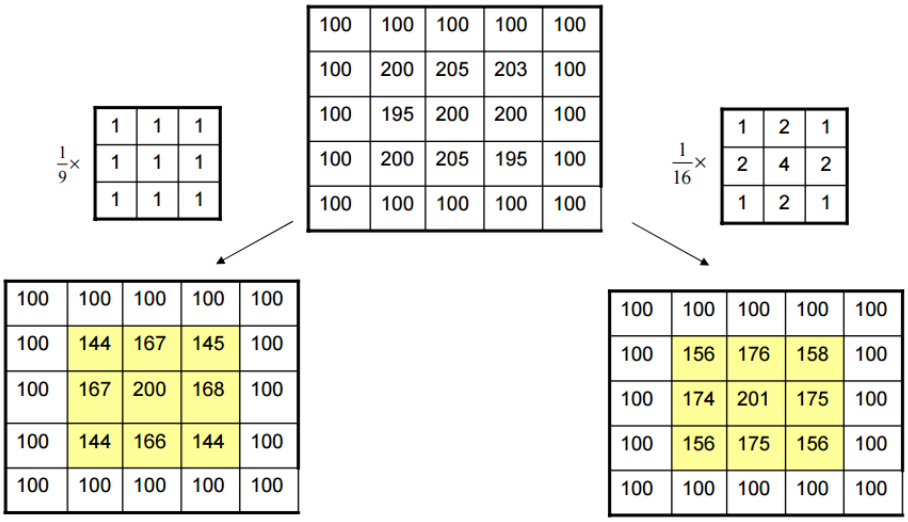


Figure (7): A numerical example for applying mean filter and weighted mean filters on a sub-image

Table (3) shows results of applying weighted mean filter on noisy images (for the three previously mentioned types of the noise) with filter size 3x3.

Table (3): Results of applying weighted mean filter on noisy images with filter size 3x3

Noise type	Noisy image	Weighted mean filter of size 3x3 for 1 times	Weighted mean filter of size 3x3 for 2 times

Salt& Pepper Noise			
Gaussian Noise			
Speckle Noise			

5. Lab. Experiments

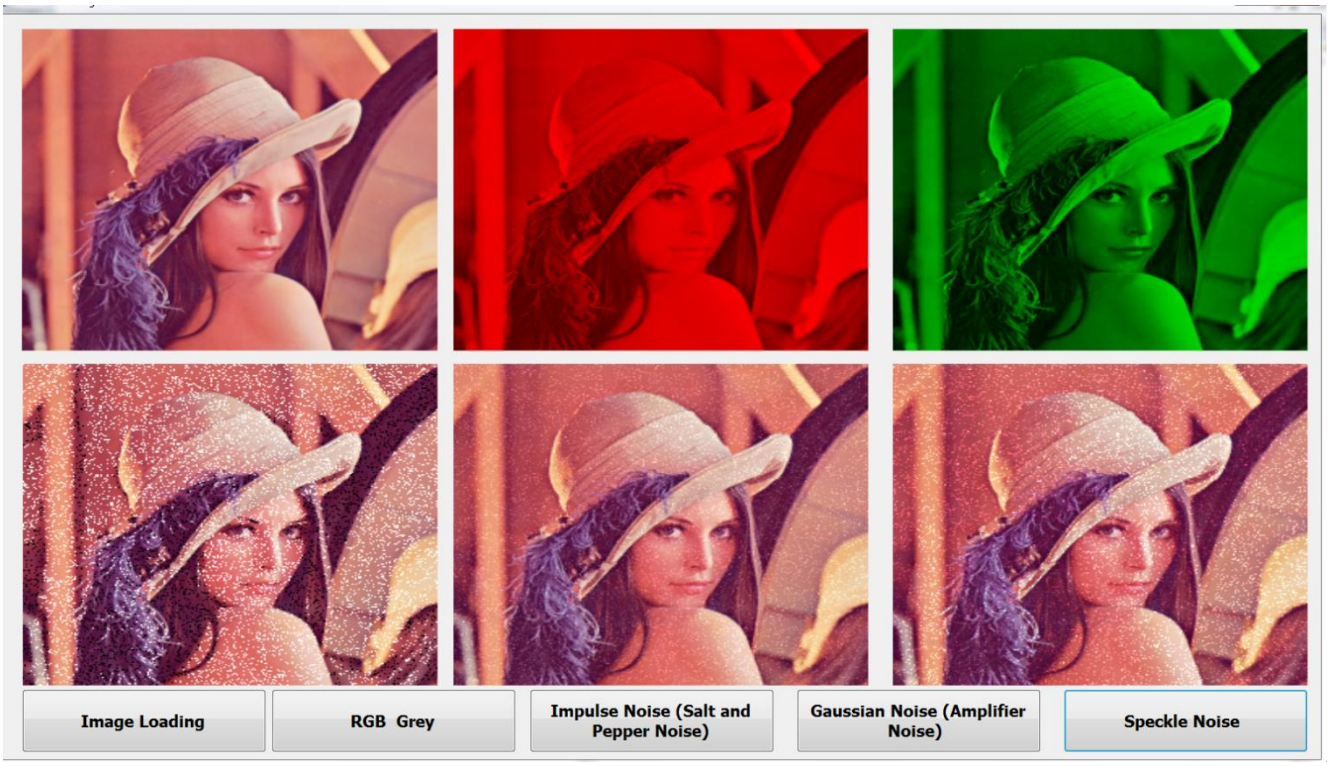
- Load a noisy true color image and try to de-noise it using linear smoothing filters:
  - Mean Filter
  - Weighted mean Filter.
- Discuss the effect of these two filters on different types of noise.

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 8 : Image Noise

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3, bmp4;
        int w, h;
        byte[,] red;
        byte[,] green;
        byte[,] blue;
        byte[,] grey;
        byte[,] Noisy_red;
        byte[,] Noisy_green;
        byte[,] Noisy_blue;
        byte Noise_T = 100;
```

```
        double Noise_Ratio = 0.15;

        public Form1()
        {
            InitializeComponent();
        }


        private void button1_Click(object sender, EventArgs e)
        {
            //openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Noisy_red = new byte[w, h];
            Noisy_green = new byte[w, h];
            Noisy_blue = new byte[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Noisy_red[i, j] = red[i, j];
                    Noisy_green[i, j] = green[i, j];
                    Noisy_blue[i, j] = blue[i, j];
                }
            }

            Random Rnd = new Random();
            int Noise_Count = (int)((w * h) * Noise_Ratio);
            for (int i = 0; i < Noise_Count; i++)
            {

                int x = (int)(Rnd.NextDouble() * w);
                int y = (int)(Rnd.NextDouble() * h);
                if (grey[x, y] < Noise_T)
                {

                    Noisy_red[x, y] = 0;
                    Noisy_green[x, y] = 0;
                    Noisy_blue[x, y] = 0;
                }
                else
                {

                    Noisy_red[x, y] = 255;
                    Noisy_green[x, y] = 255;
                    Noisy_blue[x, y] = 255;
                }
            }

            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    bmp2.SetPixel(i, j, Color.FromArgb(Noisy_red[i, j], Noisy_green[i,
j], Noisy_blue[i, j]));
                }
            }
            pictureBox4.Image = bmp2;
```

```
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;
            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];
            grey = new byte[w, h];

            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);
            bmp4 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));

                    grey[i, j] = (byte)((red[i, j] + green[i, j] + blue[i, j]) / 3.0);

                    bmp4.SetPixel(i, j, Color.FromArgb(grey[i, j], grey[i, j], grey[i,
j]));


                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;
            pictureBox5.Image = bmp4;

        }

        private void button4_Click(object sender, EventArgs e)
        {
            byte Gn = 30;

            Noisy_red = new byte[w, h];
            Noisy_green = new byte[w, h];
            Noisy_blue = new byte[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Noisy_red[i, j] = red[i, j];
                    Noisy_green[i, j] = green[i, j];
                    Noisy_blue[i, j] = blue[i, j];
                }
            }

            Random Rnd = new Random();
```

```
            int Noise_Count = (int)((w * h) * Noise_Ratio);
            for (int i = 0; i < Noise_Count; i++)
            {

                int x = (int)(Rnd.NextDouble() * w);
                int y = (int)(Rnd.NextDouble() * h);

                int v1 = Noisy_red[x, y] + Gn;
                if (v1 > 255) Noisy_red[x, y] = 255;
                else if (v1 < 0) Noisy_red[x, y] = 0;
                else Noisy_red[x, y] = (byte)v1;


                int v2 = Noisy_green[x, y] + Gn;
                if (v2 > 255) Noisy_green[x, y] = 255;
                else if (v2 < 0) Noisy_green[x, y] = 0;
                else Noisy_green[x, y] = (byte)v2;

                int v3 = Noisy_blue[x, y] + Gn;
                if (v3 > 255) Noisy_blue[x, y] = 255;
                else if (v3 < 0) Noisy_blue[x, y] = 0;
                else Noisy_blue[x, y] = (byte)v3;
            }



            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    bmp2.SetPixel(i, j, Color.FromArgb(Noisy_red[i, j], Noisy_green[i,
j], Noisy_blue[i, j]));
                }
            }
            pictureBox5.Image = bmp2;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            double Sn = 1.5;

            Noisy_red = new byte[w, h];
            Noisy_green = new byte[w, h];
            Noisy_blue = new byte[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Noisy_red[i, j] = red[i, j];
                    Noisy_green[i, j] = green[i, j];
                    Noisy_blue[i, j] = blue[i, j];
                }
            }

            Random Rnd = new Random();
            int Noise_Count = (int)((w * h) * Noise_Ratio);
            for (int i = 0; i < Noise_Count; i++)
            {

                int x = (int)(Rnd.NextDouble() * w);
                int y = (int)(Rnd.NextDouble() * h);

                double v1 = Noisy_red[x, y] * Sn;
```

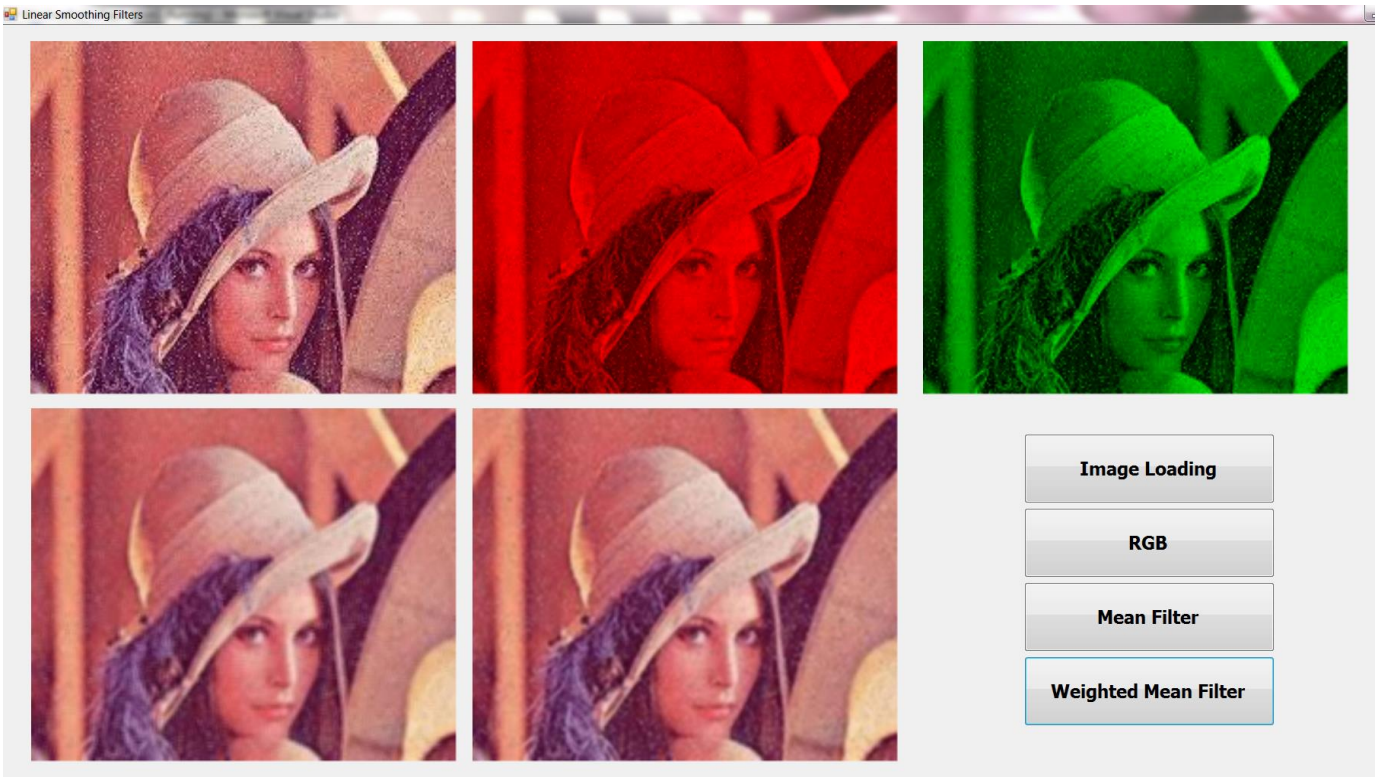
```
                if (v1 > 255) Noisy_red[x, y] = 255;
                else Noisy_red[x, y] = (byte)v1;


                double v2 = Noisy_green[x, y] * Sn;
                if (v2 > 255) Noisy_green[x, y] = 255;
                else Noisy_green[x, y] = (byte)v2;

                double v3 = Noisy_blue[x, y] * Sn;
                if (v3 > 255) Noisy_blue[x, y] = 255;
                else Noisy_blue[x, y] = (byte)v3;
            }


            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    bmp2.SetPixel(i, j, Color.FromArgb(Noisy_red[i, j], Noisy_green[i,
j], Noisy_blue[i, j]));
                }
            }
            pictureBox6.Image = bmp2;

        }
    }
}
```

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 9 : Linear Smoothing Filters

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3, bmp4;
        int w, h;
        byte[,] red;
        byte[,] green;
        byte[,] blue;
        int[,] Smoothed_red;
        int[,] Smoothed_green;
        int[,] Smoothed_blue;
```

```
        public Form1()
        {
            InitializeComponent();
        }

        public void MaskedFilters(byte[,] image, int w, int h, int[,] filter, int fs,
ref int[,] Result)
        {

            int fh = fs / 2;

            for (int x = 0; x < w; x++)
            {
                for (int y = 0; y < h; y++)
                {
                    int G = 0;

                    for (int i = -fh; i <= fh; i++)
                    {
                        int px = x + i;
                        if (px < 0)
                            px = 0;
                        if (px >= w)
                            px = w - 1;
                        for (int j = -fh; j <= fh; j++)
                        {
                            int py = y + j;
                            if (py < 0)
                                py = 0;
                            if (py >= h)
                                py = h - 1;
                            G = G + image[px, py] * filter[i + 1, j + 1];
                        }
                    }
                    Result[x, y] = G;

                }
            }

        }



        private void button1_Click(object sender, EventArgs e)
        {
            //openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            int fs = 3;

            int[,] filter = new int[3, 3] { { 1, 1, 1 }, { 1, 1, 1 }, { 1, 1, 1 } };
            Smoothed_red = new int[w, h];
            Smoothed_green = new int[w, h];
            Smoothed_blue = new int[w, h];


            MaskedFilters(red, w, h, filter, fs, ref Smoothed_red);
            MaskedFilters(green, w, h, filter, fs, ref Smoothed_green);
            MaskedFilters(blue, w, h, filter, fs, ref Smoothed_blue);
```

```
            int fsum = 9;
            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    bmp2.SetPixel(i, j, Color.FromArgb(Smoothed_red[i, j] / fsum,
Smoothed_green[i, j] / fsum, Smoothed_blue[i, j] / fsum));
                }
            }
            pictureBox4.Image = bmp2;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;
            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];


            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);
            bmp4 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));

                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;

        }

        private void button4_Click(object sender, EventArgs e)
        {
            int fs = 3;


            int[,] filter = new int[3, 3] { { 1, 2, 1 }, { 2, 4, 2 }, { 1, 2, 1 } };

            Smoothed_red = new int[w, h];
            Smoothed_green = new int[w, h];
            Smoothed_blue = new int[w, h];


            MaskedFilters(red, w, h, filter, fs, ref Smoothed_red);
```

```
            MaskedFilters(green, w, h, filter, fs, ref Smoothed_green);
            MaskedFilters(blue, w, h, filter, fs, ref Smoothed_blue);

            int fsum = 16;

            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    bmp2.SetPixel(i, j, Color.FromArgb(Smoothed_red[i, j] / fsum,
Smoothed_green[i, j] / fsum, Smoothed_blue[i, j] / fsum));
                }
            }
            pictureBox5.Image = bmp2;
        }
    }
}
```

Lab. 10: Order Smoothing Filters (Min, Max, Median, Mode, K_Mean)

1. Smoothing Order Filters

Order filters are based on ordering the pixels contained in an image by using sliding window technique to perform pixel-by-pixel operation in a filtering algorithm. The local statistics obtained from the neighborhood of the center pixel gives a lot of information about its expected value. Min, max, median, mode, K_Mean filters are examples for this type of filters.

A. Min Filter

The minimum filter selects the smallest value within the pixel values. This is accomplished by finding the minimum intensity value of all the pixels within a windowed region around the pixel. Minimum filter is useful for reducing noise of salt type.

B. Max Filter

Maximum filter selects the largest value within of pixel values. Minimum filter is useful for reducing noise of pepper type. Figure 1 shows a numerical example for min and max filters.

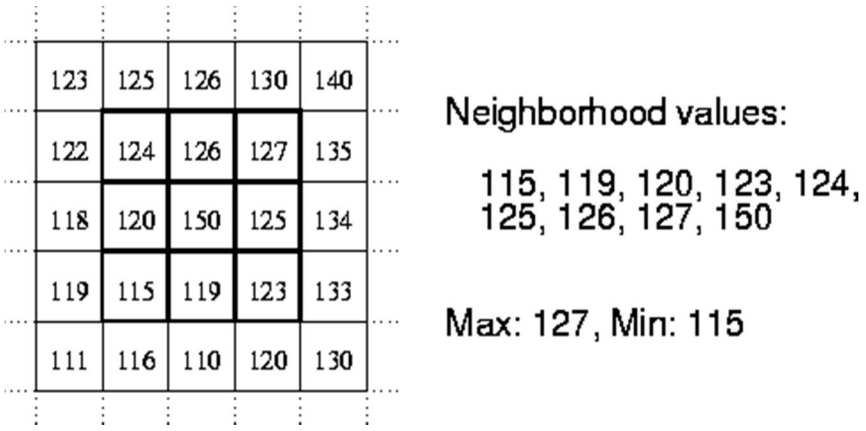


Figure (1): A numerical example for min and max filters

C. Median Filter

The median filter is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image. Since the median value must actually be the value of one of the pixels in the neighborhood, the median filter does not create new unrealistic pixel values when the filter straddles an edge. For this reason the median filter is much better at preserving sharp edges than the mean filter.

The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value as shown in figure 2.

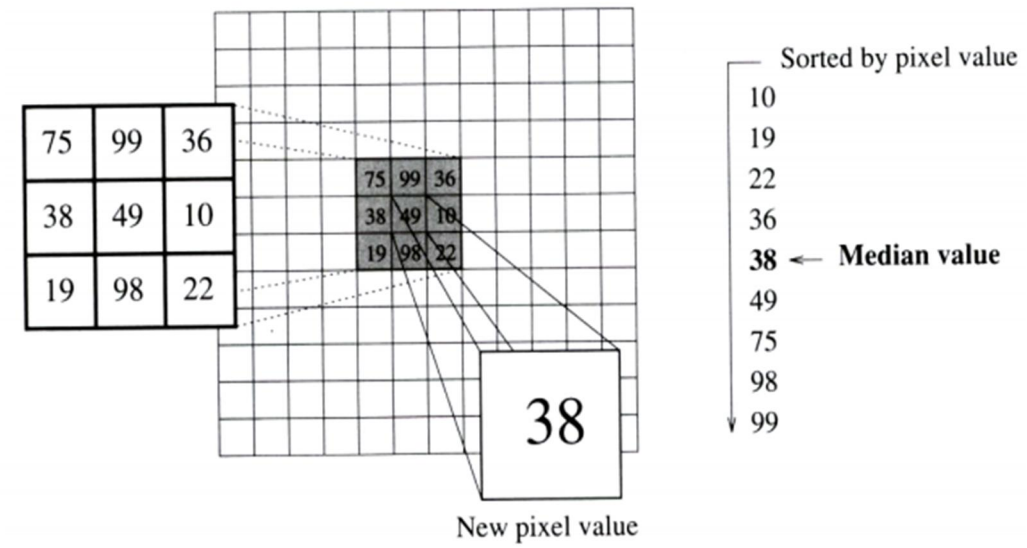


Figure (2): A numerical example for median filter

The median filter is ideal for removing impulsive noise, sometimes referred to as "salt and pepper" while preserving image detail. The main disadvantage of median filter and all types of order filter is computational complexity.

D. Mode Filter

In mode filter, each pixel value is replaced by its most common neighbor. This is a particularly useful filter for classification procedures where each pixel corresponds to an object, which must be placed into a class; in remote sensing, for example, each class could be some type of terrain, crop type, water, etc. For example, given the grey scale 3x3 pixel window:

22	77	48
150	77	158
0	77	219

Pixels based on frequency:

0 ⇨ 1
22 ⇨ 1
48 ⇨ 1
77 ⇨ 3
150 ⇨ 1
158 ⇨ 1
219 ⇨ 1

Thus, the center pixel would be left at 77 since 77 is the most frequently occurring value in the list of pixels.

E. K_Mean Filter

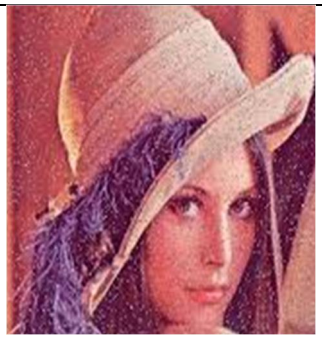
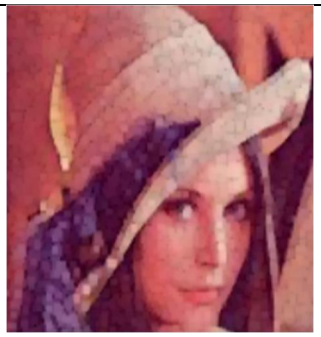
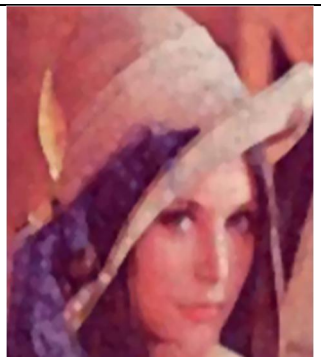
K_Mean filter works by discarding smallest and largest K values of the filter window and then taking the mean of the remaining values as a new pixel value. For the example, consider the following image segment:

100	30	89
200	220	158
0	77	5

When K_Mean filter is applied on the center value of the segment with K=2, then the two smallest values (0 and 5) and the two largest values (220 and 200) are discarded and the average of the remaining values ((30+ 77+89+100+158)/5=96) is considered as a new pixel value.

Table (1) shows the results of applying order filters on a true color noisy image with filter size 3x3 for one time.

Table (1): The results of applying order filters on true color noisy images with filter size 3x3 for one time

Type	Result
Original Noisy Image	
Min Filter	
Max Filter	
Median Filter	

Mode Filter	
K_Mean Filter (K=2)	

2. Lab. Experiments

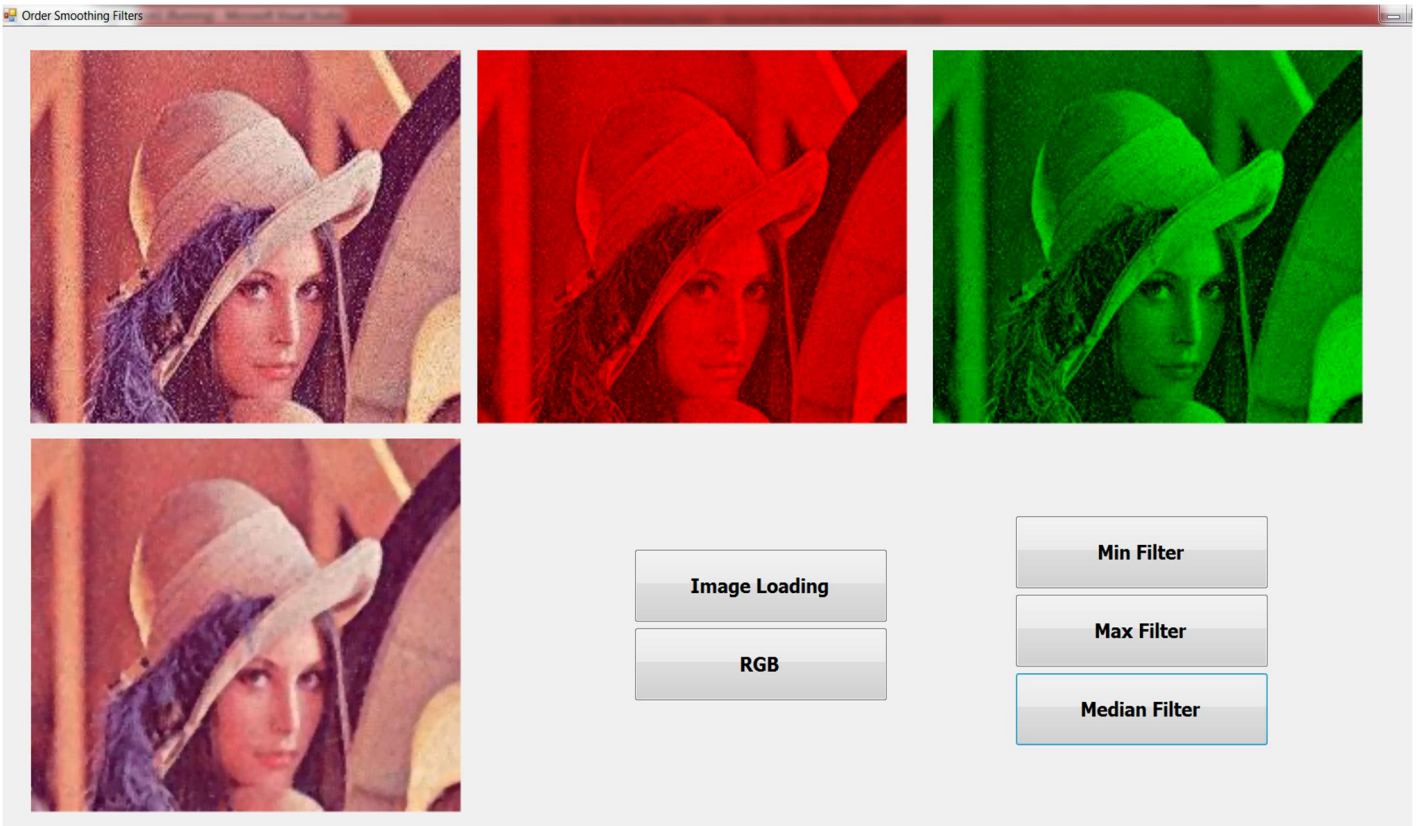
- Load a noisy true color image and try de-noising it using order smoothing filters:
  - Min Filter.
  - Max Filter
  - Median Filter.
  - Mode Filter
  - K_Mean Filter
- Discuss the effect of these filters on the different types of noise.

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 10 : Order Smoothing Filters

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3;
        int w, h;
        byte[,] red;
        byte[,] green;
        byte[,] blue;
        byte[,] Smoothed_red;
        byte[,] Smoothed_green;
        byte[,] Smoothed_blue;
```

```
        public Form1()
        {
            InitializeComponent();
        }

        public void OrderFilters(byte[,] Image, int wid, int hgt, int fs, int Pos, ref
byte[,] Min)
        {
            int fh = fs / 2;

            byte[] TempList = new byte[fs * fs];
            for (int x = 0; x < wid; x++)
            {
                for (int y = 0; y < hgt; y++)
                {
                    int s = 0;
                    for (int i = -fh; i <= fh; i++)
                    {
                        int px = x + i;
                        if (px < 0)
                            px = 0;
                        if (px >= wid)
                            px = wid - 1;
                        for (int j = -fh; j <= fh; j++)
                        {
                            int py = y + j;
                            if (py < 0)
                                py = 0;
                            if (py >= hgt)
                                py = hgt - 1;
                            TempList[s++] = Image[px, py];
                        }
                    }

                    for (int i = 0; i < (fs * fs) - 1; i++)
                        for (int j = i + 1; j < (fs * fs); j++)
                        {
                            if (TempList[i] > TempList[j])
                            {
                                byte Tv = TempList[i];
                                TempList[i] = TempList[j];
                                TempList[j] = Tv;
                            }
                        }
                    Min[x, y] = TempList[Pos];
                }
            }

        }



        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
```

```
        {
            // Min Filter

            int fs = 3;

            Smoothed_red = new byte[w, h];
            Smoothed_green = new byte[w, h];
            Smoothed_blue = new byte[w, h];



            OrderFilters(red, w, h, fs, 0, ref Smoothed_red);
            OrderFilters(green, w, h, fs, 0, ref Smoothed_green);
            OrderFilters(blue, w, h, fs, 0, ref Smoothed_blue);



            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    bmp2.SetPixel(i, j, Color.FromArgb(Smoothed_red[i, j],
Smoothed_green[i, j], Smoothed_blue[i, j]));
                }
            }
            pictureBox4.Image = bmp2;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;
            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];


            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));

                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;
```

```
        }

        private void button4_Click(object sender, EventArgs e)
        {
            // Max Filter
            int fs = 3;

            Smoothed_red = new byte[w, h];
            Smoothed_green = new byte[w, h];
            Smoothed_blue = new byte[w, h];

            OrderFilters(red, w, h, fs, (fs * fs) - 1, ref Smoothed_red);
            OrderFilters(green, w, h, fs, (fs * fs) - 1, ref Smoothed_green);
            OrderFilters(blue, w, h, fs, (fs * fs) - 1, ref Smoothed_blue);



            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    bmp2.SetPixel(i, j, Color.FromArgb(Smoothed_red[i, j],
Smoothed_green[i, j], Smoothed_blue[i, j]));
                }
            }
            pictureBox4.Image = bmp2;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            // Median Filter

            int fs = 3;

            Smoothed_red = new byte[w, h];
            Smoothed_green = new byte[w, h];
            Smoothed_blue = new byte[w, h];


            OrderFilters(red, w, h, fs, (fs * fs) / 2, ref Smoothed_red);
            OrderFilters(green, w, h, fs, (fs * fs) / 2, ref Smoothed_green);
            OrderFilters(blue, w, h, fs, (fs * fs) / 2, ref Smoothed_blue);



            bmp2 = new Bitmap(w, h);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    bmp2.SetPixel(i, j, Color.FromArgb(Smoothed_red[i, j],
Smoothed_green[i, j], Smoothed_blue[i, j]));
                }
            }
            pictureBox4.Image = bmp2;
        }
    }
}
```

Lab. 11: Gradient Filters (Prewitt, Sobel, Laplace)

1- Edge Meaning

An edge may be characterized as a set of joined pixels that forms a separation between two disjoints areas. Edge detection is fundamentally, a method of dividing an image into segments of discontinuity by locating the points, in a digital image, where there is an unexpected change in image intensities. These points are then connected together to work as boundaries for a closed object. An example for edge image is shown in Figure (1).

Figure (1): An example for edge image

2- Edge detection methods

Edge detection is the name for a set of mathematical methods which aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. Edge detection methods are divided into two main groups as follows:

(i) First Order Edge Detectors

The first order edge detection methods, also called gradient methods, detect edges by searching in the image for the minimum and maximum values of its first derivative. An object exists in the image if the gradient is above a given threshold. The most popular first order edge detection operators are

Prewitt, Roberts, and Sobel. They are all characterized by a 3x3 pattern grid, so they can be applied on the image easily and efficiently.

Sobel edge detection works on extracting all edges existing in the image, regardless of edges direction. The main advantage of Sobel operation is that it can provide both a differencing and smoothing effects. It is implemented as the summation of two directional edge enhancement operations. The resulted image will look like a unidirectional outlines of the objects existing in the original image. Regions with a constant brightness become black, while regions with changing brightness become highlighted on black background. The operator is composed of a pair of 3×3 convolution kernels, and they are expressed as:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Simply, each kernel is like the other but with 90°rotation angle. The kernels can be applied in a separate fashion upon the input image to achieve measurements of the gradient (G) component in each direction ($G_x$and G_y) separately, then the gradient magnitude can be determined using:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

and the theta (θ) that exists between Gx and Gy can be found as:

$$\theta = \tan^{-1}(\frac{G_y}{G_x})$$

The function of ***Prewitt*** edge detector is almost same as of Sobel detector but have different kernels which are:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$$

(ii) Second Order Edge Detector

In the second-order derivative edge detection techniques, some form of spatial second-order derivative is applied to highlight the edges of the image. When significant spatial change occurs then in the output of second derivative operator an edge is recorded.

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise. The operator normally takes a single gray level image as input and produces another gray level image as output. The Laplacian L(x, y) of an image with pixel intensity values I(x, y) is given by:

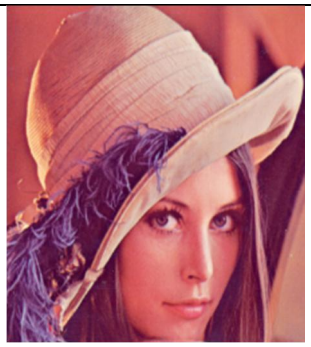
$$L(x,y) = \frac{\partial^2 I}{\partial\, x^2} + \frac{\partial^2 I}{\partial\, y^2}$$

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. The commonly used Laplace kernel is:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Table (1) shows results of applying different edge detection methods on an example image.

Table (1): Results of applying different edge detection methods on an example image

Type	Result
Original Image	
Sobel	
Prewitt	
Laplace	

3. Lab. Experiments

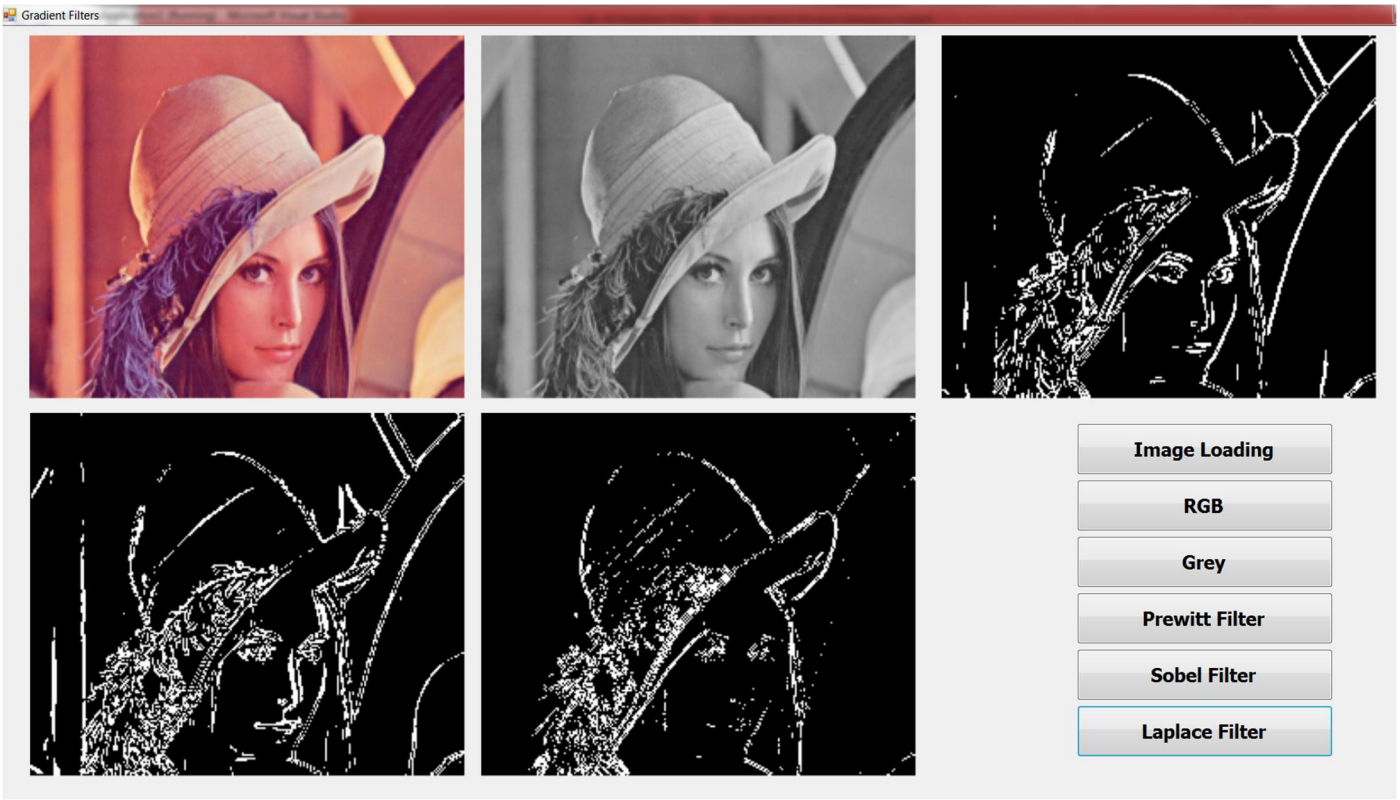
- Load a true color image and try to detect its edge map using:
  - Sobel edge detector.
  - Prewitt edge detector.
  - Laplace edge detector.
- Discuss the differences among results of these methods.

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 11 : Gradient Filters

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3;
        int w, h;
        byte[,] red;
        byte[,] green;
        byte[,] blue;
        byte[,] Edge;
        byte[,] Grey;

        public Form1()
```

```
        {
            InitializeComponent();
        }

        public void MaskedFilters(byte[,] image, int w, int h, int[,] filter, int fs,
ref int[,] Result)
        {

            int fh = fs / 2;

            for (int x = 0; x < w; x++)
            {
                for (int y = 0; y < h; y++)
                {
                    int G = 0;

                    for (int i = -fh; i <= fh; i++)
                    {
                        int px = x + i;
                        if (px < 0)
                            px = 0;
                        if (px >= w)
                            px = w - 1;
                        for (int j = -fh; j <= fh; j++)
                        {
                            int py = y + j;
                            if (py < 0)
                                py = 0;
                            if (py >= h)
                                py = h - 1;
                            G = G + image[px, py] * filter[i + 1, j + 1];
                        }
                    }
                    Result[x, y] = G;

                }
            }

        }



        private void button1_Click(object sender, EventArgs e)
        {
            //openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            // Prewitt Filter
            int[,] Gxfilter = new int[3, 3] { { -1, 0, 1 }, { -1, 0, 1 }, { -1, 0, 1 }
};
            int[,] Gyfilter = new int[3, 3] { { 1, 1, 1 }, { 0, 0, 0 }, { -1, -1, -1 }
};
            Edge = new byte[w, h];
            int[,] Gx = new int[w, h];
            int[,] Gy = new int[w, h];

            MaskedFilters(Grey, w, h, Gxfilter, 3, ref Gx);
            MaskedFilters(Grey, w, h, Gyfilter, 3, ref Gy);
```

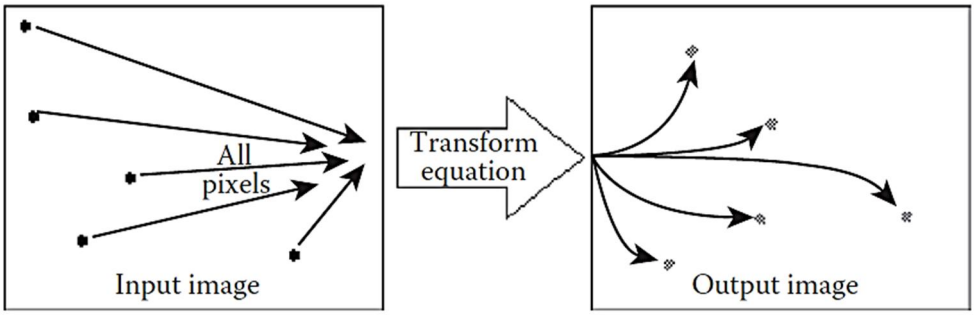
```
            bmp2 = new Bitmap(w, h);
            byte thershold = 120;
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    byte G = (byte)(Math.Sqrt((Gx[i, j] * Gx[i, j]) + (Gy[i, j] * Gy[i,
j])));

                    if (G > thershold)
                        Edge[i, j] = 255;
                    else
                        Edge[i, j] = 0;
                    bmp2.SetPixel(i, j, Color.FromArgb(Edge[i, j], Edge[i, j], Edge[i,
j]));
                }
            }
            pictureBox3.Image = bmp2;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;
            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];


            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);


            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));




                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;

        }

        private void button4_Click(object sender, EventArgs e)
        {
            // Sobel Filter
```

```
            int[,] Gxfilter = new int[3, 3] { { -1, 0, 1 }, { -2, 0, 2 }, { -1, 0, 1 }
};
            int[,] Gyfilter = new int[3, 3] { { 1, 2, 1 }, { 0, 0, 0 }, { -1, -2, -1 }
};
            Edge = new byte[w, h];
            int[,] Gx = new int[w, h];
            int[,] Gy = new int[w, h];

            MaskedFilters(Grey, w, h, Gxfilter, 3, ref Gx);
            MaskedFilters(Grey, w, h, Gyfilter, 3, ref Gy);


            bmp2 = new Bitmap(w, h);
            byte thershold = 120;
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    byte G = (byte)(Math.Sqrt((Gx[i, j] * Gx[i, j]) + (Gy[i, j] * Gy[i,
j])));

                    if (G > thershold)
                        Edge[i, j] = 255;
                    else
                        Edge[i, j] = 0;
                    bmp2.SetPixel(i, j, Color.FromArgb(Edge[i, j], Edge[i, j], Edge[i,
j]));
                }
            }
            pictureBox4.Image = bmp2;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            // Laplace Filter
            int[,] Gxfilter = new int[3, 3] { { 0, 1, 0 }, { 1, -4, 1 }, { 0, 1, 0 } };
            Edge = new byte[w, h];
            int[,] Gx = new int[w, h];

            MaskedFilters(Grey, w, h, Gxfilter, 3, ref Gx);


            bmp2 = new Bitmap(w, h);
            byte thershold = 50;
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    if (Math.Abs(Gx[i, j]) > thershold)
                        Edge[i, j] = 255;
                    else
                        Edge[i, j] = 0;
                    bmp2.SetPixel(i, j, Color.FromArgb(Edge[i, j], Edge[i, j], Edge[i,
j]));
                }
            }
            pictureBox5.Image = bmp2;
        }

        private void button6_Click(object sender, EventArgs e)
        {
            bmp1 = new Bitmap(w, h);
            Grey = new byte[w, h];
            for (int i = 0; i < w; i++)
```

```
            {
                for (int j = 0; j < h; j++)
                {
                    Grey[i, j] = (byte)((red[i, j] + green[i, j] + blue[i, j]) / 3);
                    bmp1.SetPixel(i, j, Color.FromArgb(Grey[i, j], Grey[i, j], Grey[i,
j]));
                }
            }
            pictureBox2.Image = bmp1;
        }
    }
}
```

Lab. 12 & 13 & 14: Image Transforms

A transform maps image data into a different mathematical space via a transformation equation. Basically, image transforms map the image data from the ***spatial domain to the frequency domain*** (also called the spectral domain), where all the pixels in the input (spatial domain) contribute to each value in the output (frequency domain) as shown in Figure (2).



All pixels in the input image contribute to each value in the output image for frequency transforms.

Figure (2): *Discrete transforms*

Image frequencies are important because of the following basic fact: ***Low frequencies correspond to the important image features, whereas high frequencies correspond to the details of the image, which are less important***. Thus, when a transform isolates the various image frequencies, pixels that correspond to high frequencies can be quantized heavily, while pixels that correspond to low frequencies should be quantized lightly or not at all. This is how a transform can compress an image very effectively by losing information, but only information associated with unimportant image details.

1.1 Fourier Transform

The Fourier transform is the best known, and the most widely used. It was developed by Jean Baptiste Joseph Fourier (1768–1830) to explain the distribution of temperature and heat conduction. Since that time the Fourier transform has found numerous uses,

including vibration analysis in mechanical engineering, circuit analysis in electrical engineering, and in digital image processing. The Fourier transform decomposes a complex signal into a weighted sum of a zero frequency term (the DC term that is related to the average value), and sinusoidal terms, the basic functions, where each sinusoid is a harmonic of the fundamental.

Assuming an N×N image, the equation for the 2-D discrete Fourier:

$$\boldsymbol{F(u,v) = \frac{1}{N}\sum_{r=0}^{N-1}\sum_{c=0}^{N-1} I(r,c)e^{-j2\pi\frac{(ur+vc)}{N}}}$$

The base of the natural logarithmic function e is about 2.71828; j, the imaginary coordinates for a complex number, equal $\sqrt{-1}$. The basic functions are sinusoidal in nature, as can be seen by Euler's identity:

$$\boldsymbol{e^{jx} = \cos x + j\ \sin x}$$

The Fourier transform equation can be written as:

$$\boldsymbol{F(u,v) = \frac{1}{N}\sum_{r=0}^{N-1}\sum_{c=0}^{N-1} I(r,c)\left[\cos\left(\frac{2\pi}{N}(ur+vc)\right) - j\ \sin\left(\frac{2\pi}{N}(ur+vc)\right)\right]}$$

The complex spectral component can be represented by ***F(u,v)=Re(u,v)+j Im(u,v)***, where Re(u,v) is the real part and Im(u,v) is the imaginary part. The magnitude and phase of a complex spectral component can be defined as:

$$\boldsymbol{MAGNITUDE = |F(u,v)| = \sqrt{[Re(u,v)]^2 + [Im(u,v)]^2}}$$

$$\boldsymbol{PHASE = \varphi\,(u,v) = \tan^{-1}\left[\frac{Im(u,v)}{Re(u,v)}\right]}$$

The magnitude of a sinusoid is simply its peak value, and the phase contains information about where objects are in an image. The inverse 2-D DFT is given by:

$$F^{-1}[F(u,v)] = I(r,c) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi\frac{(ur+vc)}{N}}$$

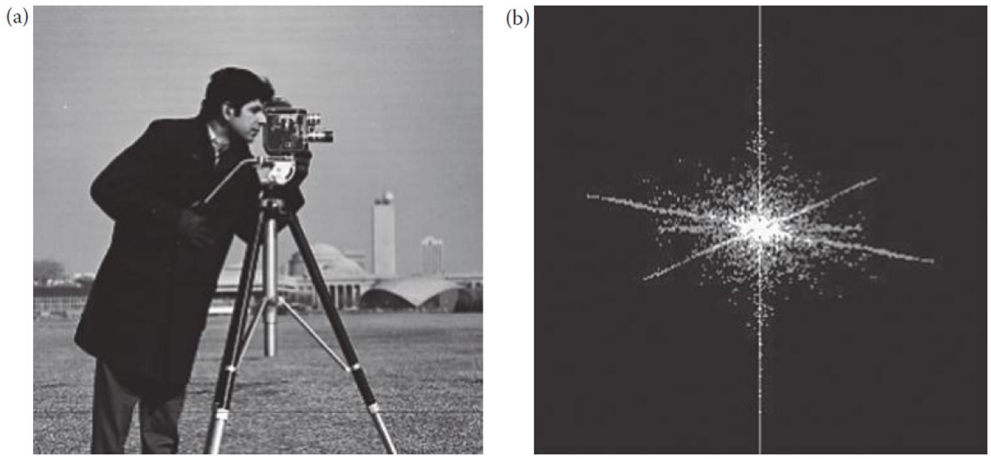


Figure (3): *(a) Original image, (b) Fourier spectrum of (a)*

2.2 Discrete Cosine Transform

The cosine transform, like the Fourier transform, uses sinusoidal basis functions. The difference is that the cosine transform basis functions are not complex; they use only ***cosine functions,*** and not sine functions. The 2-D discrete cosine transform (DCT) equation for an N x N image is given by:

$$C(u,v) = \alpha(u)\alpha(v)\sum_{r=0}^{N-1}\sum_{c=0}^{N-1} I(r,c)\cos\left[\frac{(2r+1)u\pi}{2N}\right]\cos\left[\frac{(2c+1)v\pi}{2N}\right]$$

Where:

$$\alpha(u),\alpha(v) = \begin{cases} \sqrt{\frac{1}{N}} & for\ u,v = 0 \\ \sqrt{\frac{2}{N}} & for\ u,v = 1,2,\ldots\ldots,N-1 \end{cases}$$

Since this transform uses only the cosine function it can be calculated using only real arithmetic, instead of complex arithmetic as the DFT requires. The inverse cosine transform is given by:

$$C^{-1}[C(u,v)] = I(r,c) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v)\cos\left[\frac{(2r+1)u\pi}{2N}\right]\cos\left[\frac{(2c+1)v\pi}{2N}\right]$$

The important feature of the DCT, the feature that makes it so useful in data compression, is it takes correlated input data and concentrates its energy in just the first few transform coefficients. The DCT produces the frequency spectrum F(u,v) corresponding to the spatial signal f(r,c) in which the first few DCT coefficients is the DC coefficient of f(r,c) and the other DCT coefficients represent the various changing AC components of f(r,c). Compressing data with the DCT is therefore done by quantizing the coefficients. The small ones are quantized coarsely (possibly all the way to zero), and the large ones can be quantized finely to the nearest integer. Decompression is done by performing the inverse DCT on the quantized coefficients. This results in data items that are not identical to the original ones but are not much different. Figure (4) shows an example image of applying DCT on an image and Figure (5) shows a numerical example of DCT result when applied on 8x8 image block.

Figure (4): DCT example

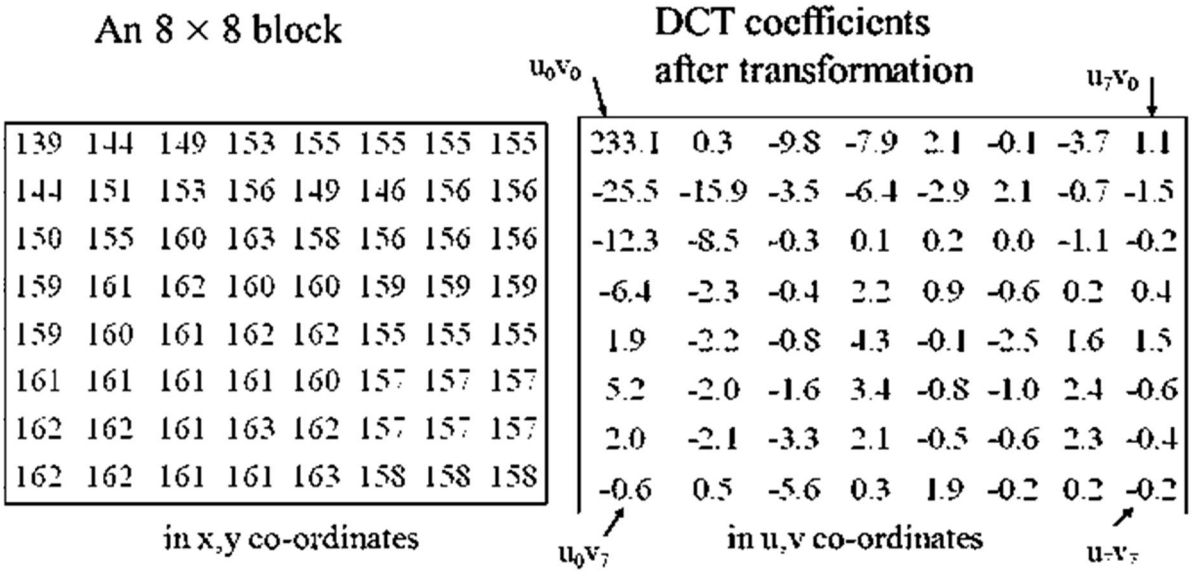


Figure (5): A numerical DCT example

2.3 Haar Wavelet Transform

The objective of the wavelet transform is to decompose the input image into components that are easier to deal with, have special interpretations, or have some components that can be thresholded away, for compression purposes.

Consider Haar wavelet transform that replaces the original sequence with its pairwise average $X_{n,i}$ and difference $d_{n,j}$ defined as follows:

$$\boldsymbol{X_{n,i} = \frac{X_{n,2i} + X_{n,2i+1}}{\sqrt{2}}}, \qquad \boldsymbol{d_{n,i} = \frac{X_{n,2i} - X_{n,2i+1}}{\sqrt{2}}}$$

The averages and differences are applied only on consecutive pairs of input sequences whose first element has an even index. Therefore, the number of elements in each set $\{X_{n,i}\}$ and $\{d_{n,i}\}$ is exactly half of the number of elements in the original sequence. Form a new sequence having length equal to that of the original sequence by concatenating the two sequences. It is easily verified that the original sequence can be reconstructed from the transformed sequence using the relations:

$$\boldsymbol{X_{n,2i} = X_{n,i} + d_{n,i}}, \qquad \boldsymbol{X_{n,2i+1} = X_{n,i} - d_{n,i}}$$

For an NxN input image, the two-dimensional DWT proceeds as follows:

1. Convolve each row of the image with $h_0[n]$ and $h_1[n]$, discard the odd numbered columns of the resulting arrays, and concatenate them to form a transformed row.

2. After all rows have been transformed, convolve each column of the result with $h_0[n]$ and $h_1[n]$. Again discard the odd numbered rows and concatenate the result.

By applying DWT, the image is actually decomposed into ***four sub bands***. These four sub bands arise from separable applications of vertical and horizontal filters. The sub-bands labeled ***Low-High (LH), High-Low (HL) and High-High (HH)*** represent the finest scale wavelet coefficients (i.e., detail images) while the ***sub-band Low-Low (LL)*** corresponds to coarse level coefficients (i.e., approximation image). To obtain the next coarse level of wavelet coefficients, the sub band LL alone is further decomposed and critically sampled in iterative manner. This process continues until some final scale is reached. This leads to two- level wavelet decomposition are shown in Figure (6).

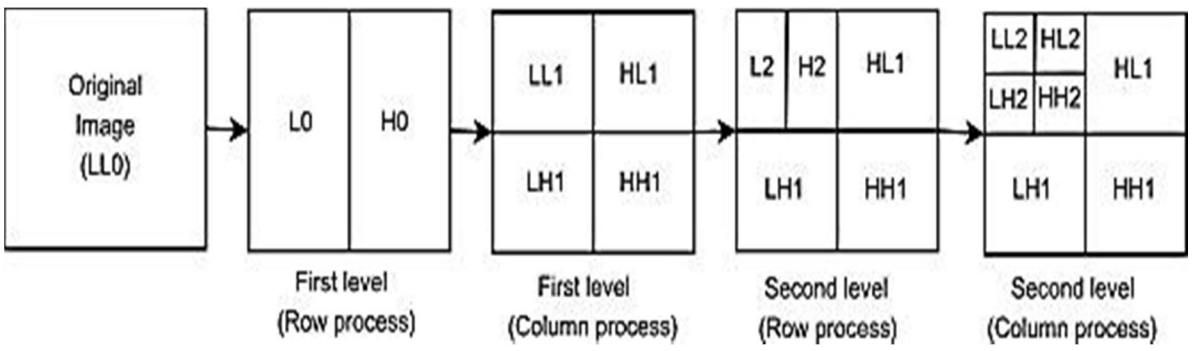


Figure (6): *Two- level wavelet decomposition*

Some advantages of wavelet transform:

a) One of the main advantages of wavelets is that they offer a simultaneous localization in time and frequency domain.

b) The second main advantage of wavelets is that, using fast wavelet transform, it is computationally very fast.

c) Wavelets have the great advantage of being able to separate the fine details in the image.

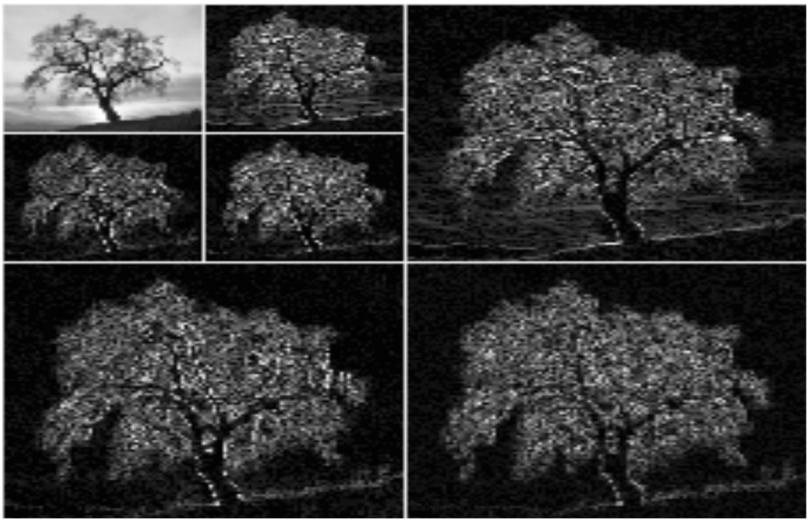


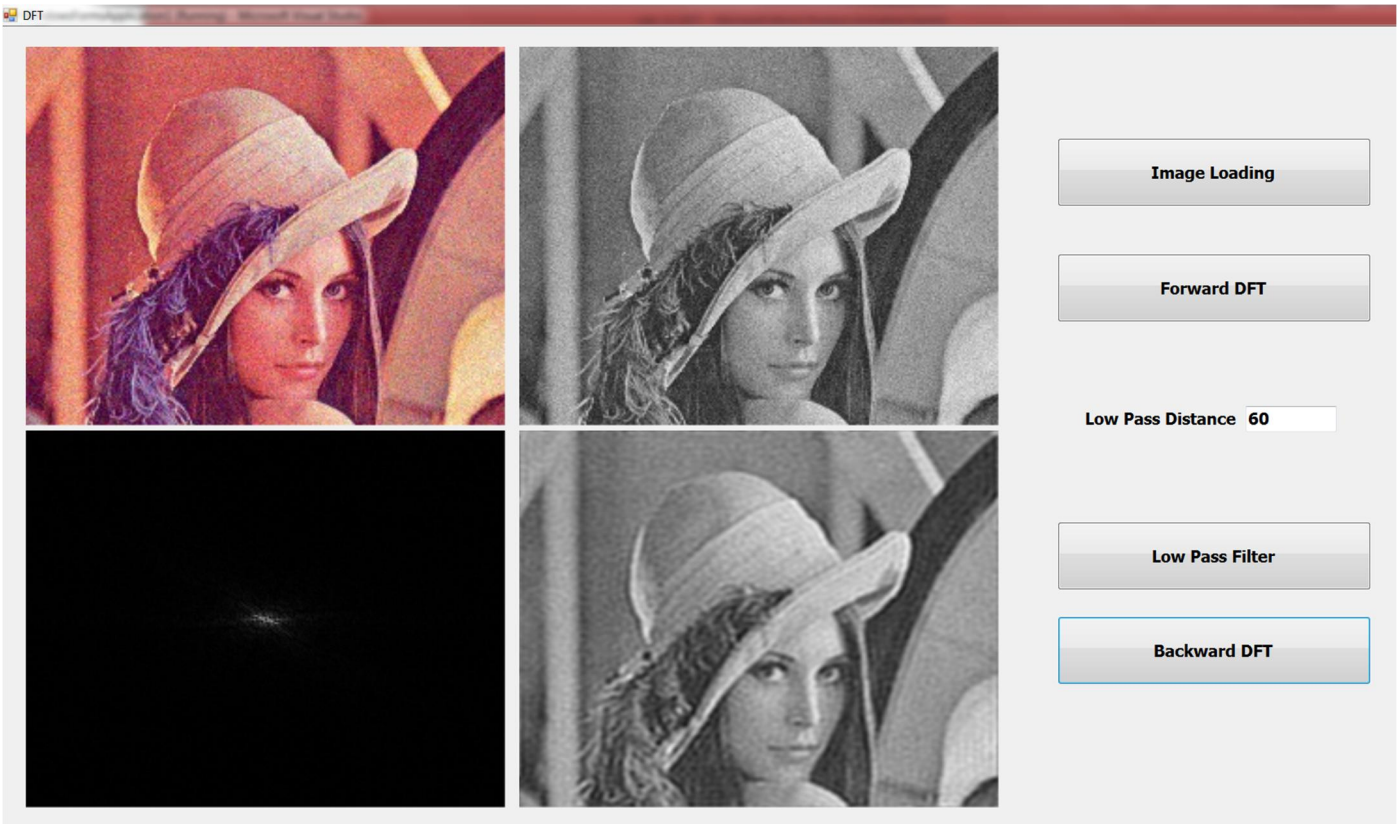
Figure (7): *An example image of Haar wavelet transform*

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 12 : Discrete Fourier Transform (DFT)

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp2;
        int W, H;
        byte[,] Grey;
        double[,] DFTR, DFTImg;

        double T;
        public Form1()
        {
```

```
            InitializeComponent();
        }

        private void ForwardDFT(double[] Ir, double[] Img, int N, ref double[] Fr, ref
double[] FImg)
        {
            Fr = new double[N];
            FImg = new double[N];
            for (int u = 0; u < N; u++)
            {
                for (int v = 0; v < N; v++)
                {
                    double c = Math.Cos(T * u * v);
                    double s = Math.Sin(T * u * v);
                    Fr[u] = Fr[u] + Ir[v] * c + Img[v] * s;
                    FImg[u] = FImg[u] - Ir[v] * s + Img[v] * c;

                }

                Fr[u] = Fr[u] / N;
                FImg[u] = FImg[u] / N;
            }
        }

        private void BackwardDFT(double[] Ir, double[] Img, int N, ref double[] Fr, ref
double[] FImg)
        {
            Fr = new double[N];
            FImg = new double[N];
            for (int u = 0; u < N; u++)
            {
                for (int v = 0; v < N; v++)
                {
                    double c = Math.Cos(T * u * v);
                    double s = Math.Sin(T * u * v);
                    Fr[u] = Fr[u] + Ir[v] * c - Img[v] * s;
                    FImg[u] = FImg[u] + Ir[v] * s + Img[v] * c;

                }
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;

            W = bmp.Width;
            H = bmp.Height;

            Grey = new byte[W, H];
            bmp2 = new Bitmap(W, H);

            for (int i = 0; i < W; i++)
            {
                for (int j = 0; j < H; j++)
                {
                    Color p = bmp.GetPixel(i, j);
                    Grey[i, j] = (byte)((p.R + p.G + p.B) / 3);
                    bmp2.SetPixel(i, j, Color.FromArgb(Grey[i, j], Grey[i, j], Grey[i,
j]));
                }
            }
```

```
        pictureBox2.Image = bmp2;



    }

    private void button2_Click(object sender, EventArgs e)
    {
        bmp2 = new Bitmap(W, H);

        DFTR = new double[W, H];
        DFTImg = new double[W, H];

        // Vertical DFT
        T = (2 * Math.PI) / W;
        for (int j = 0; j < H; j++)
        {
            double[] T1 = new double[W];
            double[] T2 = new double[W];

            //Copy Col. Pixels
            for (int i = 0; i < W; i++)
            {
                T1[i] = Math.Pow(-1, (i + j)) * Grey[i, j];
                T2[i] = 0;
            }

            //Apply Forward DFT
            double[] RT = new double[W];
            double[] IT = new double[W];
            ForwardDFT(T1, T2, W, ref RT, ref IT);
            for (int i = 0; i < W; i++)
            {
                DFTR[i, j] = RT[i];
                DFTImg[i, j] = IT[i];

            }
        }


        // Horizontal DFT
        T = (2 * Math.PI) / H;
        for (int i = 0; i < W; i++)
        {
            double[] T1 = new double[H];
            double[] T2 = new double[H];

            //Copy row Pixels
            for (int j = 0; j < H; j++)
            {
                T1[j] = DFTR[i, j];
                T2[j] = DFTImg[i, j];
            }

            //Apply ForwardDFT
            double[] RT = new double[H];
            double[] IT = new double[H];
            ForwardDFT(T1, T2, H, ref RT, ref IT);
            for (int j = 0; j < H; j++)
            {
                DFTR[i, j] = RT[j];
                DFTImg[i, j] = IT[j];
```

```
                }

            }


            //DFT Magnitude Drawing

            for (int i = 0; i < W; i++)
            {
                for (int j = 0; j < H; j++)
                {
                    byte M = (byte)(Math.Sqrt((DFTR[i, j] * DFTR[i, j]) + (DFTImg[i, j]
* DFTImg[i, j])) * 20);
                    bmp2.SetPixel(i, j, Color.FromArgb(M, M, M));
                }
            }

            pictureBox3.Image = bmp2;

        }

        private void button3_Click(object sender, EventArgs e)
        {
            bmp2 = new Bitmap(W, H);



            // Vertical DFT
            T = (2 * Math.PI) / W;
            for (int j = 0; j < H; j++)
            {
                double[] T1 = new double[W];
                double[] T2 = new double[W];

                //Copy Col. Pixels
                for (int i = 0; i < W; i++)
                {
                    T1[i] = DFTR[i, j];
                    T2[i] = DFTImg[i, j];
                }

                //Apply Backward DFT
                double[] RT = new double[W];
                double[] IT = new double[W];

                BackwardDFT(T1, T2, W, ref RT, ref IT);


                for (int i = 0; i < W; i++)
                {
                    DFTR[i, j] = RT[i];
                    DFTImg[i, j] = IT[i];

                }
            }


            // Horizontal DFT
            T = (2 * Math.PI) / H;
            for (int i = 0; i < W; i++)
            {
                double[] T1 = new double[H];
                double[] T2 = new double[H];
```

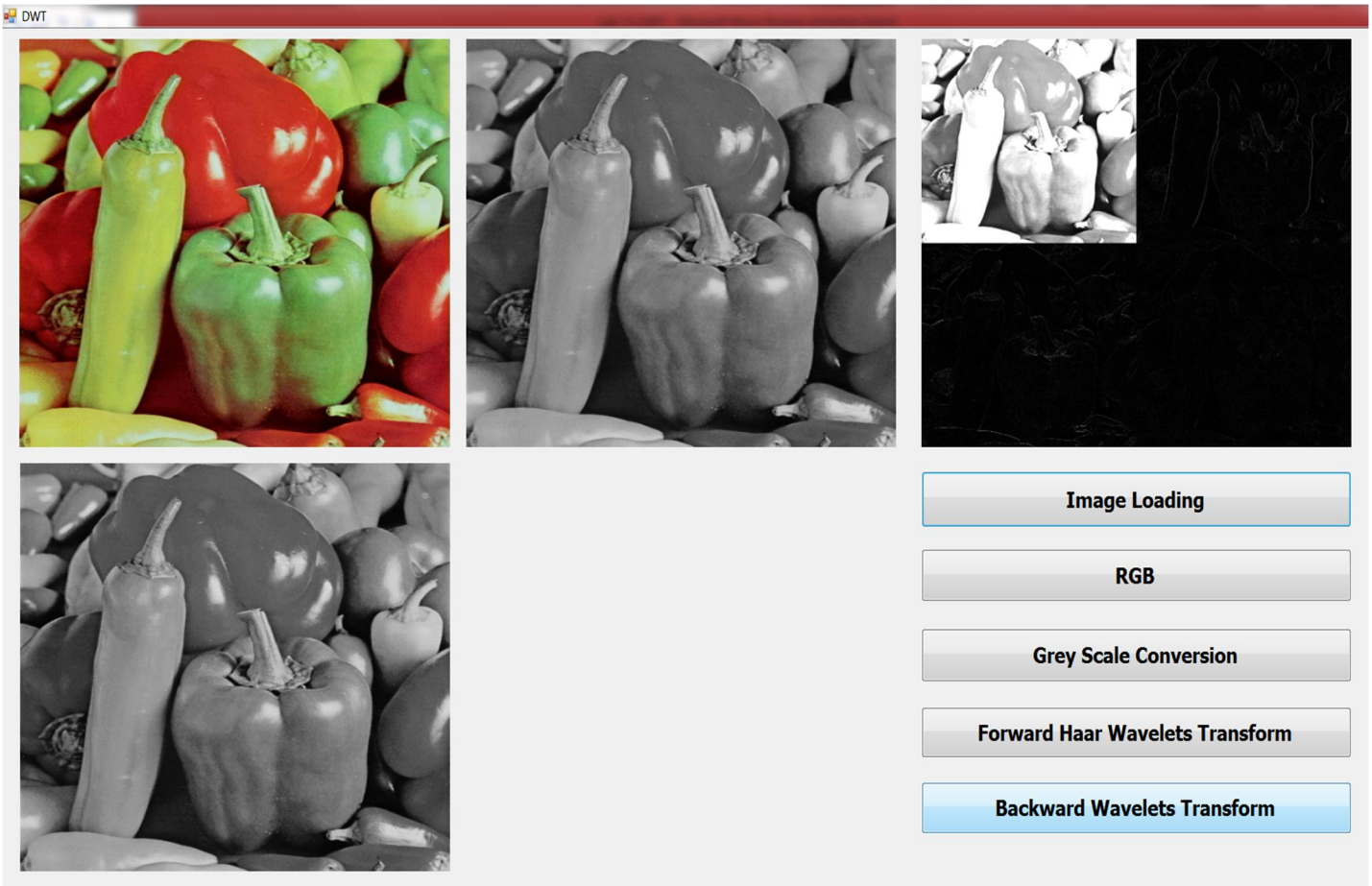
```
                //Copy row Pixels
                for (int j = 0; j < H; j++)
                {
                    T1[j] = DFTR[i, j];
                    T2[j] = DFTImg[i, j];
                }

                //Apply Backward DFT
                double[] RT = new double[H];
                double[] IT = new double[H];

                BackwardDFT(T1, T2, H, ref RT, ref IT);

                for (int j = 0; j < H; j++)
                {
                    DFTR[i, j] = RT[j];
                    DFTImg[i, j] = IT[j];

                }

            }


            //DFT Magnitude Drawing

            for (int i = 0; i < W; i++)
            {
                for (int j = 0; j < H; j++)
                {
                    byte M = (byte)(Math.Sqrt((DFTR[i, j] * DFTR[i, j]) + (DFTImg[i, j]
* DFTImg[i, j])));
                    bmp2.SetPixel(i, j, Color.FromArgb(M, M, M));
                }
            }

            pictureBox4.Image = bmp2;
        }

        private void button4_Click(object sender, EventArgs e)
        {

            // Low Pass Filter
            double D0 = Convert.ToDouble(textBox1.Text);

            for (int i = 0; i < W; i++)
            {
                for (int j = 0; j < H; j++)
                {
                    double Hv = 0;
                    double D = Math.Sqrt(((i - (W / 2)) * (i - W / 2)) + ((j - (H / 2))
* (j - (H / 2))));
                    if (D <= D0) Hv = 1; else Hv = 0;
                    DFTR[i, j] = DFTR[i, j] * Hv;
                    DFTImg[i, j] = DFTImg[i, j] * Hv;

                }
            }

        }
    }
}
```

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 13 : Discrete Wavelets Transform (DWT)

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2, bmp3;
        int w, h;
        byte[,] red, green, blue, Grey;
        double[,] fhaarwavelet, bhaarwavelet;
```

```
        public Form1()
        {
            InitializeComponent();
        }

        public void ForwardHarr(byte[,] I, int Wid, int Hgt, ref double[,] Result)
        {
            int i, j, t1, t2;
            double[,] Temp = new double[w, h];
            double TwoS = 2.0;//Math.Sqrt(2);

            for (j = 0; j < Hgt; j++)  //Horizontal DWT
            {
                t1 = 0;
                t2 = Wid / 2;
                for (i = 0; i < Wid; i = i + 2)
                {
                    if (i + 1 < Wid)
                    {
                        Temp[t1, j] = ((I[i, j] + I[i + 1, j]) / TwoS);
                        Temp[t2, j] = (((I[i, j] - I[i + 1, j])) / TwoS);
                        t1 = t1 + 1;
                        t2 = t2 + 1;
                    }
                    else
                    {
                        Temp[t1, j] = ((I[i, j] + I[i, j]) / TwoS);
                        Temp[t2, j] = (((I[i, j] - I[i, j])) / TwoS);
                    }
                }
            }
            for (i = 0; i < Wid; i++)    //Vertical DWT
            {
                t1 = 0;
                t2 = Hgt / 2;
                for (j = 0; j < Hgt; j = j + 2)
                {
                    if (j + 1 < Hgt)
                    {
                        Result[i, t1] = ((Temp[i, j] + Temp[i, j + 1]) / TwoS);

                        Result[i, t2] = (((Temp[i, j] - Temp[i, j + 1])) / TwoS);

                        t1 = t1 + 1;
                        t2 = t2 + 1;
                    }
                    else
                    {
                        Result[i, t1] = ((Temp[i, j] + Temp[i, j]) / TwoS);
                        Result[i, t2] = (((Temp[i, j] - Temp[i, j])) / TwoS);
                    }
                }
            }
        }

        public void BackwardHarr(double[,] I, int Wid, int Hgt, ref double[,] Result)
        {
            int i, j, t1, t2;
            double[,] Temp = new double[w, h];

            for (i = 0; i < Wid; i++)
            {
                t1 = 0;
                t2 = Hgt / 2;
```

```
            for (j = 0; j < Hgt; j = j + 2)
            {
                if (j + 1 < Hgt)
                {
                    Temp[i, j] = ((I[i, t1] + I[i, t2]));

                    Temp[i, j + 1] = (((I[i, t1] - I[i, t2])));

                    t1 = t1 + 1;
                    t2 = t2 + 1;
                }
                else
                {
                    Temp[i, j] = ((I[i, t1] + I[i, t2]));

                }
            }
        }
        for (j = 0; j < Hgt; j++)
        {
            t1 = 0;
            t2 = Wid / 2;
            for (i = 0; i < Wid; i = i + 2)
            {
                if (i + 1 < Wid)
                {
                    Result[i, j] = ((Temp[t1, j] + Temp[t2, j]));
                    Result[i + 1, j] = (((Temp[t1, j] - Temp[t2, j])));
                    t1 = t1 + 1;
                    t2 = t2 + 1;

                }
                else
                {
                    Result[i, j] = ((Temp[t1, j] + Temp[t2, j]));

                }
            }
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        //openFileDialog1.Filter = "Jpeg Images|*.jpg|Bmp Images|*.bmp";
        openFileDialog1.ShowDialog();
        bmp = new Bitmap(openFileDialog1.FileName);
        pictureBox1.Image = bmp;
    }

    private void button2_Click(object sender, EventArgs e)
    {
        bmp1 = new Bitmap(w, h);
        fhaarwavelet = new double[w, h];
        ForwardHarr(Grey, w, h, ref fhaarwavelet);
        for (int i = 0; i < w; i++)
        {
            for (int j = 0; j < h; j++)
            {
                double p = Math.Abs(fhaarwavelet[i, j])*2;
                if (p > 255)
                    p = 255;
                bmp1.SetPixel(i, j, Color.FromArgb((int)p, (int)p, (int)p));
            }
        }
```

```
            pictureBox3.Image = bmp1;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            w = bmp.Width;
            h = bmp.Height;
            red = new byte[w, h];
            green = new byte[w, h];
            blue = new byte[w, h];

            bmp1 = new Bitmap(w, h);
            bmp2 = new Bitmap(w, h);
            bmp3 = new Bitmap(w, h);

            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Color p = bmp.GetPixel(i, j);

                    red[i, j] = p.R;
                    bmp1.SetPixel(i, j, Color.FromArgb(red[i, j], 0, 0));

                    green[i, j] = p.G;
                    bmp2.SetPixel(i, j, Color.FromArgb(0, green[i, j], 0));

                    blue[i, j] = p.B;
                    bmp3.SetPixel(i, j, Color.FromArgb(0, 0, blue[i, j]));
                }
            }
            pictureBox2.Image = bmp1;
            pictureBox3.Image = bmp2;
            pictureBox4.Image = bmp3;

        }

        private void button4_Click(object sender, EventArgs e)
        {

            bmp1 = new Bitmap(w, h);
            Grey = new byte[w, h];
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    Grey[i, j] = (byte)((red[i, j] + green[i, j] + blue[i, j]) / 3);
                    bmp1.SetPixel(i, j, Color.FromArgb(Grey[i, j], Grey[i, j], Grey[i,
j]));
                }
            }
            pictureBox2.Image = bmp1;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            bmp1 = new Bitmap(w, h);
            bhaarwavelet = new double[w, h];
            BackwardHarr(fhaarwavelet, w, h, ref bhaarwavelet);
            for (int i = 0; i < w; i++)
            {
                for (int j = 0; j < h; j++)
                {
                    double p = Math.Abs(bhaarwavelet[i, j]);
```

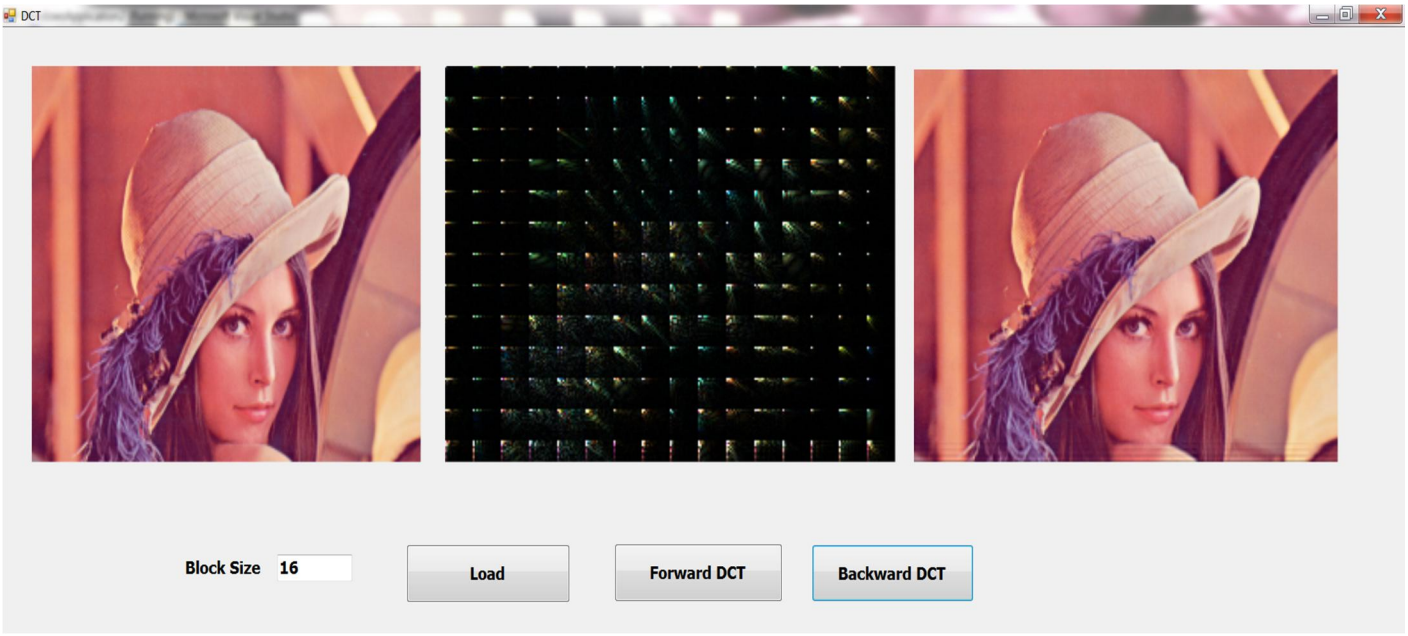
```
                    if (p > 255)
                        p = 255;
                    bmp1.SetPixel(i, j, Color.FromArgb((int)p, (int)p, (int)p));
                }
            }
            pictureBox4.Image = bmp1;
        }
    }
}
```

University of Baghdad / College of Science
Department of Computer Science

Image Processing Lab.
Third Class / Morning Study

Lab. 14 : Discrete Cosine Transform (DCT)

Design



Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace WindowsApplication2
{
    public partial class Form1 : Form
    {
        Bitmap bmp, bmp1, bmp2;
        int[,] red;
        int[,] green;
        int[,] blue;
        double[,] Fr, Fg, Fb;
        double[,] Br, Bg, Bb;
        double basev;
        double BSdouble;
        int w, h, BS, XS, YS;
        const double Pi = Math.PI;

        public double c(int u)
        {
            if (u == 0) return (1.0 / Math.Sqrt(2)); else return 1.0;
        }
```

```
        public void ReadBlock()
        {
            for (int x = XS; x < XS + BS; x++)
            {
                if (x >= w) break;
                for (int y = YS; y < YS + BS; y++)
                {
                    if (y >= h) break;
                    Color p = bmp.GetPixel(x, y);
                    red[x, y] = p.R;
                    green[x, y] = p.G;
                    blue[x, y] = p.B;
                }
            }
        }

        public void FDCT()
        {
            double sumr = 0, sumg = 0, sumb = 0;
            for (int u = 0; u < BS; u++)
            {
                double fu = c(u);
                if ((u + XS >= w)) break;
                for (int v = 0; v < BS; v++)
                {
                    if ((v + YS >= h)) break;

                    sumr = 0;
                    sumg = 0;
                    sumb = 0;

                    for (int x = 0; x < BS; x++)
                    {
                        if ((x + XS >= w)) break;
                        double p1 = Math.Cos(((2 * x + 1) * u * Pi) / BSdouble);

                        for (int y = 0; y < BS; y++)
                        {
                            if ((y + YS >= h)) break;
                            double p2 = Math.Cos(((2 * y + 1) * v * Pi) / BSdouble);
                            sumr = sumr + p1 * p2 * red[x + XS, y + YS];
                            sumg = sumg + p1 * p2 * green[x + XS, y + YS];
                            sumb = sumb + p1 * p2 * blue[x + XS, y + YS];

                        }
                    }

                    double fv = c(v);
                    Fr[u + XS, v + YS] = sumr * fu * fv * basev;
                    Fg[u + XS, v + YS] = sumg * fu * fv * basev;
                    Fb[u + XS, v + YS] = sumb * fu * fv * basev;
                    int rr = (int)Math.Abs(Fr[u + XS, v + YS]);
                    int gg = (int)Math.Abs(Fg[u + XS, v + YS]);
                    int bb = (int)Math.Abs(Fb[u + XS, v + YS]);

                    if (rr > 255) rr = 255;
                    if (gg > 255) gg = 255;
                    if (bb > 255) bb = 255;

                    bmp1.SetPixel(u + XS, v + YS, Color.FromArgb(rr, gg, bb));

                }
            }
        }
```

```
        public void BDCT()
        {
            double sumr = 0;
            double sumg = 0;
            double sumb = 0;
            for (int x = 0; x < BS; x++)
            {
                if ((x + XS >= w)) break;
                for (int y = 0; y < BS; y++)
                {
                    if ((y + YS >= h)) break;
                    sumr = 0;
                    sumg = 0;
                    sumb = 0;
                    for (int u = 0; u < BS; u++)
                    {
                        if ((u + XS >= w)) break;
                        double fu = c(u);
                        double p1 = Math.Cos(((2 * x + 1) * u * Pi) / BSdouble);
                        for (int v = 0; v < BS; v++)
                        {
                            if ((v + YS >= h)) break;
                            double fv = c(v);
                            double p2 = Math.Cos(((2 * y + 1) * v * Pi) / BSdouble);
                            sumr = sumr + p1 * p2 * Fr[u + XS, v + YS] * fu * fv;
                            sumg = sumg + p1 * p2 * Fg[u + XS, v + YS] * fu * fv;
                            sumb = sumb + p1 * p2 * Fb[u + XS, v + YS] * fu * fv;
                        }
                    }

                    Br[x + XS, y + YS] = Math.Abs(sumr * basev);
                    Bg[x + XS, y + YS] = Math.Abs(sumg * basev);
                    Bb[x + XS, y + YS] = Math.Abs(sumb * basev);
                    if (Br[x + XS, y + YS] > 255) Br[x + XS, y + YS] = 255;
                    if (Bg[x + XS, y + YS] > 255) Bg[x + XS, y + YS] = 255;
                    if (Bb[x + XS, y + YS] > 255) Bb[x + XS, y + YS] = 255;

                    bmp2.SetPixel(x + XS, y + YS, Color.FromArgb((int)(Br[x + XS, y +
YS]), (int)(Bg[x + XS, y + YS]), (int)(Bb[x + XS, y + YS])));

                }
            }

        }
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.ShowDialog();
            bmp = new Bitmap(openFileDialog1.FileName);
            pictureBox1.Image = bmp;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            BS = Convert.ToInt16(textBox1.Text);

            w = bmp.Width; h = bmp.Height;
            bmp1 = new Bitmap(w, h);
```

```
            bmp2 = new Bitmap(w, h);
            red = new int[w, h];
            green = new int[w, h];
            blue = new int[w, h];
            Fr = new double[w, h];
            Fg = new double[w, h];
            Fb = new double[w, h];
            basev = 2 / Math.Sqrt(BS * BS);
            BSdouble = (double)(2 * BS);


            for (XS = 0; XS < w; XS = XS + BS)
            {
                for (YS = 0; YS < h; YS = YS + BS)
                {
                    ReadBlock();
                    FDCT();
                }
            }
            pictureBox2.Image = bmp1;


        }

        private void button3_Click(object sender, EventArgs e)
        {
            Br = new double[w, h];
            Bg = new double[w, h];
            Bb = new double[w, h];
            for (XS = 0; XS < w; XS = XS + BS)
            {
                for (YS = 0; YS < h; YS = YS + BS)
                {
                    BDCT();

                }
            }
            pictureBox3.Image = bmp2;
        }
    }
}
```