Ministerul Educaţiei din Republica Moldova

Universitatea Liberă Internaţională din Moldova

Facultatea Informatică şi Inginerie

Catedra Tehnologii Informaţionale şi Inginerie

**RAPORT**

la lucrarea de laborator № 6

Disciplina: Prelucrarea semnalelor

*"*  *Signals filtering on the base of 2D convolution "*

**A efectuat**

**studentul gr.IA-33 Semnătura Cusnariov Ruslan**

**A verificat**

**Dr.hab., prof.univ Semnătura Perju Veaceslav**

**Chişinău 2015**

2. **Describe the algorithm and the software.**

***Listingul programului:***

using System;

using System.Collections.Generic;

using System.Text;

using System.Drawing;

using System.Drawing.Imaging;

using System.Drawing.Drawing2D;

namespace Low\_Pas

{

/// Definim datele pentru tipul Complex N=R+Ii

struct COMPLEX

{

public double real, imag;

public COMPLEX(double x, double y)

{

real = x;

imag = y;

}

public float Magnitude()

{

return ((float)Math.Sqrt(real \* real + imag \* imag));

}

public float Phase()

{

if (imag != 0 && real != 0)

{

return ((float)Math.Atan(imag / real));

}

return 0;

}

public static COMPLEX operator \*(COMPLEX x, COMPLEX y)

{

COMPLEX result = new COMPLEX(x.real\*y.real - x.imag\*y.imag, x.real\*y.imag + x.imag\*y.real);

return result;

}

}

class FFT

{

public Bitmap Obj;

public Bitmap FourierPlot;

public Bitmap PhasePlot;

public int[,] GreyImage;

public float[,] FourierMagnitude;

public float[,] FourierPhase;

float[,] FFTLog;

float[,] FFTPhaseLog;

public int[,] FFTNormalized;

public int[,] FFTPhaseNormalized;

int nx, ny; //Numarul de pixeli(inaltime si latime)

int Width, Height;

COMPLEX[,] Fourier;

public COMPLEX[,] FFTShifted;

public COMPLEX[,] Output;

public COMPLEX[,] FFTNormal;

public COMPLEX[,] FFTFiltered;

/// Constructorul parametrizat pentru FFT - Input Bitmap si Greyscale Array

public FFT(Bitmap Input)

{

Obj = Input;

Width = nx = Input.Width;

Height= ny = Input.Height;

ReadImage();

}

/// Constructorul parametrizat pentru FFT

/// <param name="Input">Greyscale Array</param>

///

public FFT(int[,] Input)

{

GreyImage = Input;

Width = nx = Input.GetLength(0);

Height = ny = Input.GetLength(1);

}

/// Constructorul pentru FT inversata

/// <param name="Input"></param>

public FFT(COMPLEX[,] Input)

{

nx = Width = Input.GetLength(0);

ny = Height = Input.GetLength(1);

Fourier = Input;

}

/// <summary>

/// Functia de prelucrare a imaginii Bitmap cu greyscale Array

/// </summary>

private void ReadImage()

{

int i, j;

GreyImage = new int[Width, Height]; //[Rinduri, coloane]

Bitmap image = Obj;

Color clr = new Color();

for (i = 0; i < image.Height; i++)

{

for (j = 0; j < image.Height; j++)

{

clr = image.GetPixel(i, j);

GreyImage[i, j] = (byte)(((byte)clr.R + (byte)clr.G + (byte)clr.B)/3);

}

}

return;

}

public Bitmap Displayimage()

{

int i, j;

Bitmap image = new Bitmap(Width, Height);

Color clr = new Color();

for (i = 0; i < image.Height; i++)

{

for (j = 0; j < image.Height; j++)

{

if (GreyImage[i,j] > 255)

GreyImage[i,j] = 255;

clr = Color.FromArgb((byte)GreyImage[i, j], (byte)GreyImage[i, j], (byte)GreyImage[i, j]);

image.SetPixel(i, j, clr);

}

}

return image;

}

public Bitmap Displayimage(int[,] image)

{

int i, j;

Bitmap output = new Bitmap(image.GetLength(0), image.GetLength(1));

Color clr = new Color();

for (i = 0; i < output.Height; i++)

{

for (j = 0; j < output.Height; j++)

{

if (image[i,j] > 255)

image[i,j] = 255;

clr = Color.FromArgb((byte)image[i, j], (byte)image[i, j], (byte)image[i, j]);

output.SetPixel(i, j, clr);

}

}

return output;;

}

public void ForwardFFT()

{

int i,j;

Fourier =new COMPLEX [Width,Height];

Output = new COMPLEX[Width, Height];

//Copierea datelor

for (i=0;i<=Width -1;i++)

for (j = 0; j <= Height - 1; j++)

{

Fourier[i, j].real =(double) GreyImage[i, j];

Fourier[i, j].imag = 0;

}

Output= FFT2D( Fourier, nx, ny, 1);

return;

}

public void FFTShift()

{

int i, j;

FFTShifted = new COMPLEX[nx, ny];

for(i=0; i<=(nx/2)-1; i++)

for (j = 0; j <= (ny / 2) - 1; j++)

{

FFTShifted[i + (nx / 2), j + (ny / 2)] = Output[i, j];

FFTShifted[i, j] = Output[i + (nx / 2), j + (ny / 2)];

FFTShifted[i + (nx / 2), j] = Output[i , j + (ny / 2)];

FFTShifted[i, j + (nx / 2)] = Output[i + (nx / 2), j ];

}

return;

}

public void RemoveFFTShift()

{

int i, j;

FFTNormal = new COMPLEX[nx, ny];

for (i = 0; i <= (nx / 2) - 1; i++)

for (j = 0; j <= (ny / 2) - 1; j++)

{

FFTNormal[i + (nx / 2), j + (ny / 2)] = FFTShifted[i, j];

FFTNormal[i, j] = FFTShifted[i + (nx / 2), j + (ny / 2)];

FFTNormal[i + (nx / 2), j] = FFTShifted[i, j + (ny / 2)];

FFTNormal[i, j + (nx / 2)] = FFTShifted[i + (nx / 2), j];

}

return;

}

public void RemoveFFTShift(COMPLEX [,] fftShifted)

{

int i, j;

FFTNormal = new COMPLEX[nx, ny];

for (i = 0; i <= (nx / 2) - 1; i++)

for (j = 0; j <= (ny / 2) - 1; j++)

{

FFTNormal[i + (nx / 2), j + (ny / 2)] = fftShifted[i, j];

FFTNormal[i, j] = fftShifted[i + (nx / 2), j + (ny / 2)];

FFTNormal[i + (nx / 2), j] = fftShifted[i, j + (ny / 2)];

FFTNormal[i, j + (nx / 2)] = fftShifted[i + (nx / 2), j];

}

return;

}

public void FFTPlot(COMPLEX[,]Output)

{

int i, j;

float max;

FFTLog = new float[nx, ny];

FFTPhaseLog = new float[nx, ny];

FourierMagnitude = new float[nx, ny];

FourierPhase = new float[nx, ny];

FFTNormalized = new int[nx, ny];

FFTPhaseNormalized = new int[nx, ny];

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

FourierMagnitude[i, j] = Output[i, j].Magnitude();

FourierPhase[i, j] = Output[i, j].Phase();

FFTLog[i, j] = (float)Math.Log(1 + FourierMagnitude[i, j]);

FFTPhaseLog[i, j] = (float)Math.Log(1 + Math.Abs(FourierPhase[i, j]));

}

//Generarea magnitudinei imaginei Bitmap

max = FFTLog[0, 0];

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

if (FFTLog[i, j] > max)

max = FFTLog[i, j];

}

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

FFTLog[i, j] = FFTLog[i, j] / max;

}

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

FFTNormalized[i, j] = (int)(2000 \* FFTLog[i, j]);

if(FFTNormalized[i, j] >= 255)

{

FFTPhaseNormalized[i, j] = 254;

}

}

//Transferarea imaginii in FFT

FourierPlot = Displayimage(FFTNormalized);

//Generarea fazei imaginei

FFTPhaseLog[0, 0] = 0;

max = FFTPhaseLog[1, 1];

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

if (FFTPhaseLog[i, j] > max)

max = FFTPhaseLog[i, j];

}

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

FFTPhaseLog[i, j] = FFTPhaseLog[i, j] / max;

}

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

FFTPhaseNormalized[i, j] = (int)(255 \* FFTPhaseLog[i, j]);

}

PhasePlot = Displayimage(FFTPhaseNormalized);

}

/// <summary>

/// generarea FFT pentru afisarea

/// </summary>

public void FFTPlot()

{

int i, j;

float max;

FFTLog = new float [nx,ny];

FFTPhaseLog = new float[nx, ny];

FourierMagnitude = new float[nx, ny];

FourierPhase = new float[nx, ny];

FFTNormalized = new int[nx, ny];

FFTPhaseNormalized = new int[nx, ny];

for(i=0;i<=Width-1;i++)

for (j = 0; j <= Height-1; j++)

{

FourierMagnitude[i, j] = Output[i, j].Magnitude();

FourierPhase[i, j] = Output[i, j].Phase();

FFTLog[i, j] = (float)Math.Log(1 + FourierMagnitude[i, j]);

FFTPhaseLog[i, j] = (float)Math.Log(1 + Math.Abs(FourierPhase[i, j]));

}

max = FFTLog[0, 0];

for(i=0;i<=Width-1;i++)

for (j = 0; j <= Height-1; j++)

{

if (FFTLog[i, j] > max)

max = FFTLog[i, j];

}

for(i=0;i<=Width-1;i++)

for (j = 0; j <= Height-1; j++)

{

FFTLog[i, j] = FFTLog[i, j] / max;

}

for(i=0;i<=Width-1;i++)

for (j = 0; j <= Height-1; j++)

{

FFTNormalized [i,j]=(int)(1000\*FFTLog[i,j]);

}

FourierPlot = Displayimage(FFTNormalized);

max = FFTPhaseLog[0, 0];

for (i = 0; i <= Width-1; i++)

for (j = 0; j <= Height-1; j++)

{

if (FFTPhaseLog[i, j] > max)

max = FFTPhaseLog[i, j];

}

for (i = 0; i <= Width-1; i++)

for (j = 0; j <= Height-1; j++)

{

FFTPhaseLog[i, j] = FFTPhaseLog[i, j] / max;

}

for (i = 0; i <= Width-1; i++)

for (j = 0; j <= Height-1; j++)

{

FFTPhaseNormalized[i, j] = (int)(2000 \* FFTLog[i, j]);

}

PhasePlot = Displayimage(FFTPhaseNormalized);

public void InverseFFT()

{

int i, j;

Output =new COMPLEX [nx,ny];

Output = FFT2D(Fourier, nx, ny, -1);

Obj = null;

//Copierea imaginei

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

GreyImage[i, j] = (int)Output[i, j].Magnitude();

}

Obj = Displayimage(GreyImage);

return;

}

public void InverseFFT(COMPLEX [,] Fourier)

{

int i, j;

//Chemarea functiei

Output = new COMPLEX[nx, ny];

Output = FFT2D(Fourier, nx, ny, -1);

for (i = 0; i <= Width - 1; i++)

for (j = 0; j <= Height - 1; j++)

{

GreyImage[i, j] = (int)Output[i, j].Magnitude();

}

Obj = Displayimage(GreyImage);

return;

}

/// Aceasta metoda implementeaza metoda Low-pass:

/// P(x,y) = F(x,y)\*H(x,y)= F{F[F(x,y)]F[H(x,y)]} =

/// = F{F(u,v)H(u,v)}

/// sintaxa

/// <param name="nFreqX"></param>

/// <param name="nFreqY"></param>

public void ApplyLowPassFilter(int nFreqX, int nFreqY)

{

int i,j;

int Cu, Cv;

COMPLEX Zero = new COMPLEX(0,0);

FFTFiltered = new COMPLEX[Width, Height];

// Implementarea coordonatelor

Cu = Width/2 - nFreqX;

Cv = Height/2 - nFreqY;

for (i = 0; i < Width ; i++)

{

for(j = 0; j < Height ; j++)

{

if (((i < Cu) && (j < Cv)) ||

((i > (Width - 1) - Cu) && (j < Cv)) ||

((i < Cu) && (j > (Height - 1) - Cv)) ||

((i > (Width - 1) - Cu)&&(j > (Height - 1) - Cv)))

{

FFTFiltered[i, j] = Fourier[i, j]; // F(u,v)\*1

}

else

FFTFiltered[i, j] = Zero; // F(u,v)\*0

}

}

}

public COMPLEX [,] FFT2D(COMPLEX[,] c, int nx, int ny, int dir)

{

int i,j;

int m;

double []real;

double []imag;

COMPLEX [,] output;//=nou COMPLEX [nx,ny];

output = c;

// Transformarea in coloane

real = new double[nx] ;

imag = new double[nx];

for (j=0;j<ny;j++)

{

for (i=0;i<nx;i++)

{

real[i] = c[i,j].real;

imag[i] = c[i,j].imag;

}

// Chemarea 1D FFT din coloane

m = (int)Math.Log((double)nx, 2);//Gasirea puterii 2 pentru numerele si punctele (ex.) nx=512 m=9

FFT1D(dir,m,ref real,ref imag);

for (i=0;i<nx;i++)

{

// c[i,j].real = real[i];

// c[i,j].imag = imag[i];

output[i, j].real = real[i];

output[i, j].imag = imag[i];

}

}

// Transformarea coloanelor

real = new double[ny];

imag = new double[ny];

for (i=0;i<nx;i++)

{

for (j=0;j<ny;j++)

{

//real[j] = c[i,j].real;

//imag[j] = c[i,j].imag;

real[j] = output[i, j].real;

imag[j] = output[i, j].imag;

}

m = (int)Math.Log((double)ny, 2);

FFT1D(dir,m,ref real,ref imag);

for (j=0;j<ny;j++)

{

//c[i,j].real = real[j];

//c[i,j].imag = imag[j];

output[i, j].real = real[j];

output[i, j].imag = imag[j];

}

}

return(output);

}

private void FFT1D(int d**i**r, int m, ref double[] x, ref double[] y )

{

long nn, i, i1, j, k, i2, l, l1, l2;

double c1, c2, tx, ty, t1, t2, u1, u2, z;

/\* Calcularea numarului de puncte \*/

nn = 1;

for (i = 0; i < m; i++)

nn \*= 2;

i2 = nn >> 1;

j = 0;

for (i = 0; i < nn - 1; i++)

{

if (i < j)

{

tx = x[i];

ty = y[i];

x[i] = x[j];

y[i] = y[j];

x[j] = tx;

y[j] = ty;

}

k = i2;

while (k <= j)

{

j -= k;

k >>= 1;

}

j += k;

}

/\* Compunerea FFT \*/

c1 = -1.0;

c2 = 0.0;

l2 = 1;

for (l = 0; l < m; l++)

{

l1 = l2;

l2 <<= 1;

u1 = 1.0;

u2 = 0.0;

for (j = 0; j < l1; j++)

{

for (i = j; i < nn; i += l2)

{

i1 = i + l1;

t1 = u1 \* x[i1] - u2 \* y[i1];

t2 = u1 \* y[i1] + u2 \* x[i1];

x[i1] = x[i] - t1;

y[i1] = y[i] - t2;

x[i] += t1;

y[i] += t2;

}

z = u1 \* c1 - u2 \* c2;

u2 = u1 \* c2 + u2 \* c1;

u1 = z;

}

c2 = Math.Sqrt((1.0 - c1) / 2.0);

if (dir == 1)

c2 = -c2;

c1 = Math.Sqrt((1.0 + c1) / 2.0);

}

/\* \*/

if (dir == 1)

{

for (i = 0; i < nn; i++)

{

x[i] /= (double)nn;

y[i] /= (double)nn;

}

}

// functia de returnare ;

return;

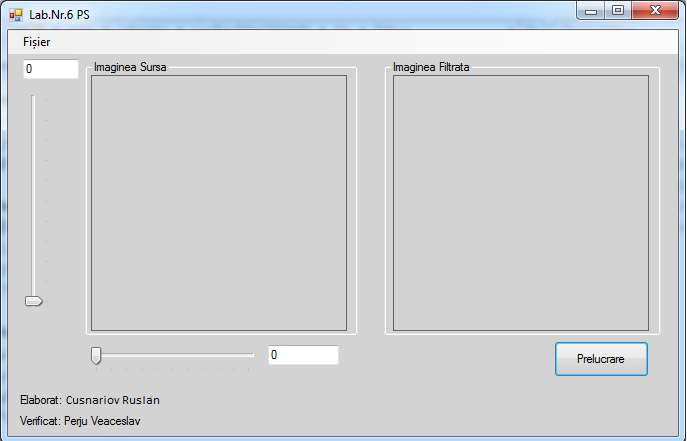
}

}

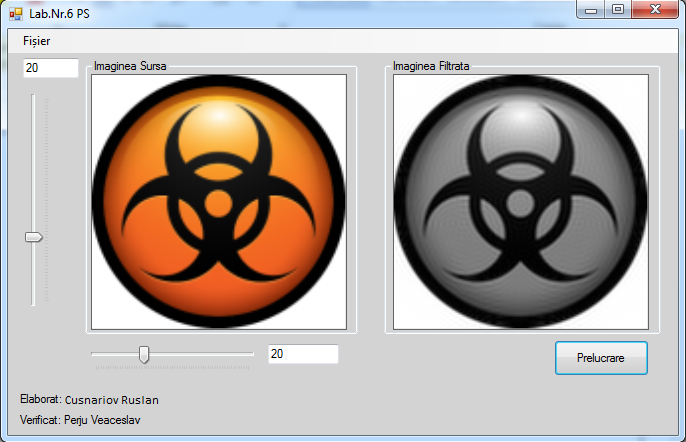
}

***Rezultatele executării:***

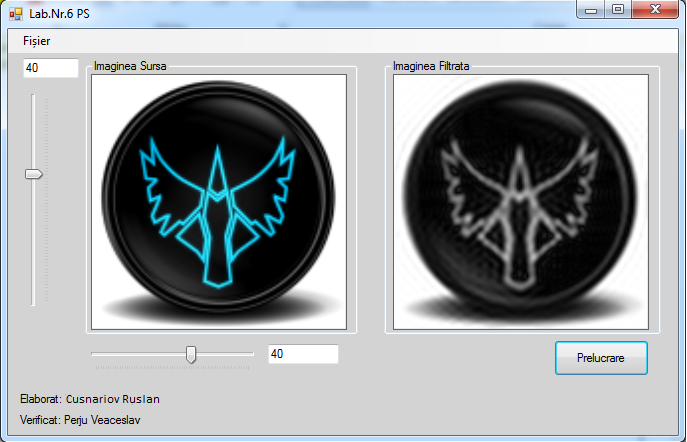
***Fig.1 – Fereastra principală a programului***

******

***Fig.2- Prelucrarea imaginii***

******

***Fig.3 - Prelucrarea altei imagini cu alți parametri***

******

**Concluzii:**