**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине **«Основы цифровой обработки изображений»**

на тему: «Методы пространственной фильтрации изображений»

Выполнил: студент гр. ИТ-31

Рябиков М.А.

Принял: преподаватель

Цалко И.Н.

Гомель 2019

**Цель:** рассмотреть возможности применения высокочастотных и низкочастотных пространственных фильтров для улучшения цифровых изображений.

1. **Задание**
2. В соответствии с вариантом реализовать низкочастотный фильтр для изображения в оттенках серого (см. табл. 1), вывести преобразованное изображение, сравнить с исходным.
3. Реализовать высокочастотную фильтрацию на основе оператора, соответствующего варианту (см. табл. 1).
4. На основе результатов выполнения задания 2 реализовать фильтрацию исходного изображения с **подъемом высоких частот.** Вывести преобразованное изображение, сравнить с исходным и с результатом по заданию 2.

|  |  |  |
| --- | --- | --- |
| Вариант | Сглаживающий фильтр | Оператор для высокочастотного фильтра |
| 6(21) | Минимума по окрестности 5×5 | Перекрестный градиентный оператор Робертса |

**Ход работы.**

Окно программы при запуске:

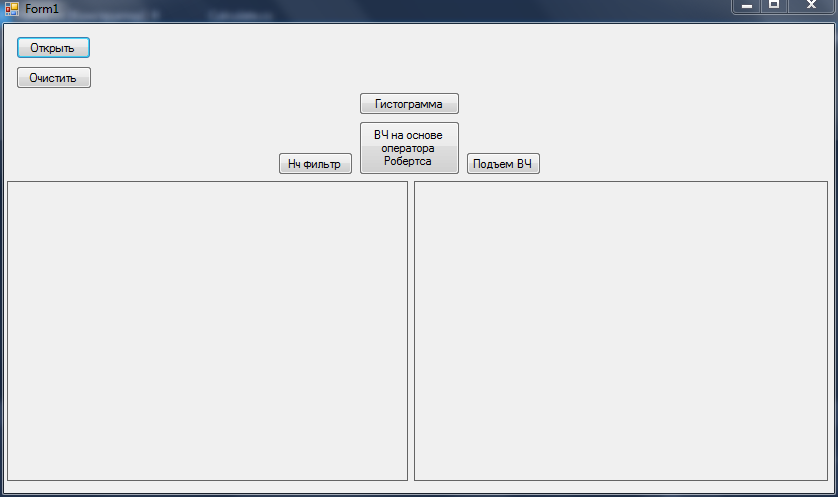


Рисунок 1 – Окно программы

Окно программы после высокочастотной фильтрации:

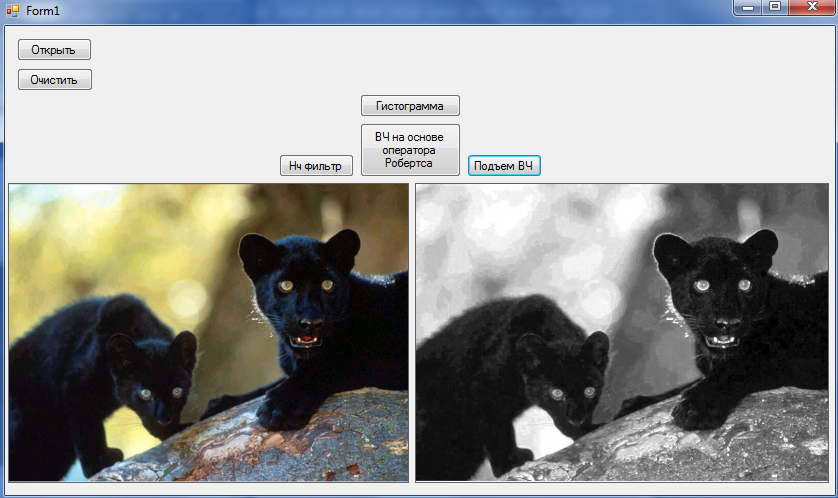


Рисунок 2 – Высококачественная фильтрация

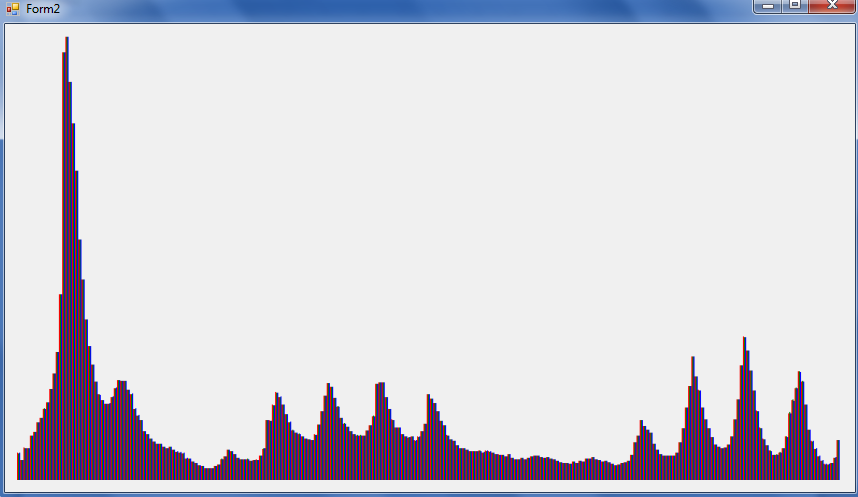


Рисунок 3 – гистограмма высококачественной фильтрации

Окно программы после низкочастотной фильтрации:

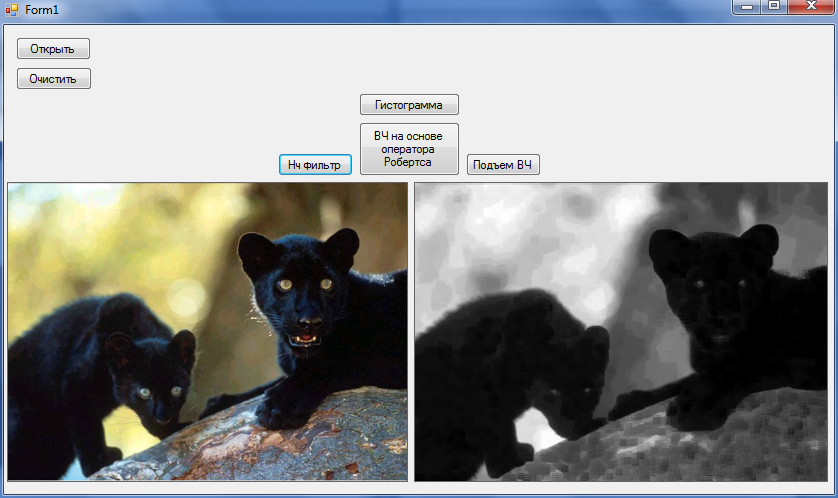


Рисунок 4 – низкокачественная фильтрация

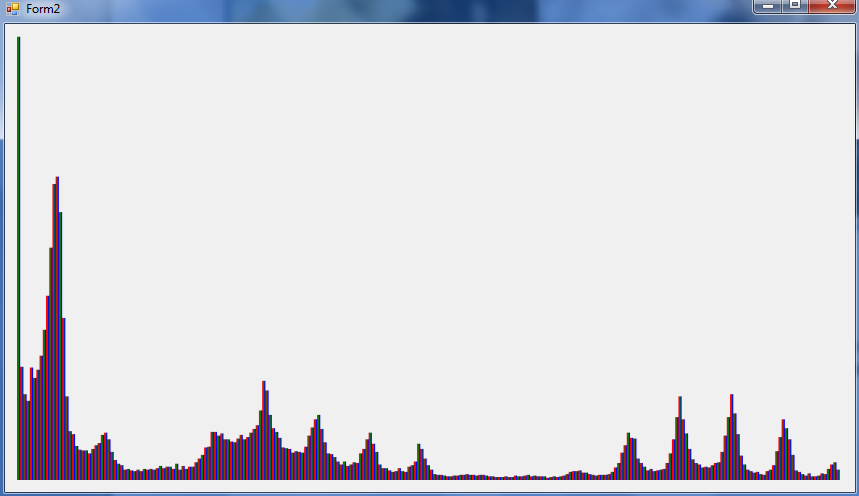


Рисунок 5 – гистограмма низкокачественной фильтрации

Окно программы после высококачественной фильтрации на основе оператора Робертса:

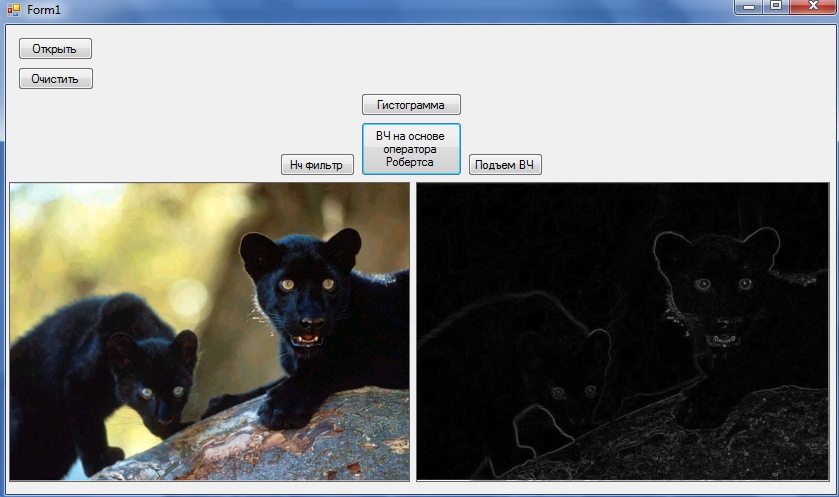


Рисунок 6 – Высококачественная фильтрация на основе оператора Робертса

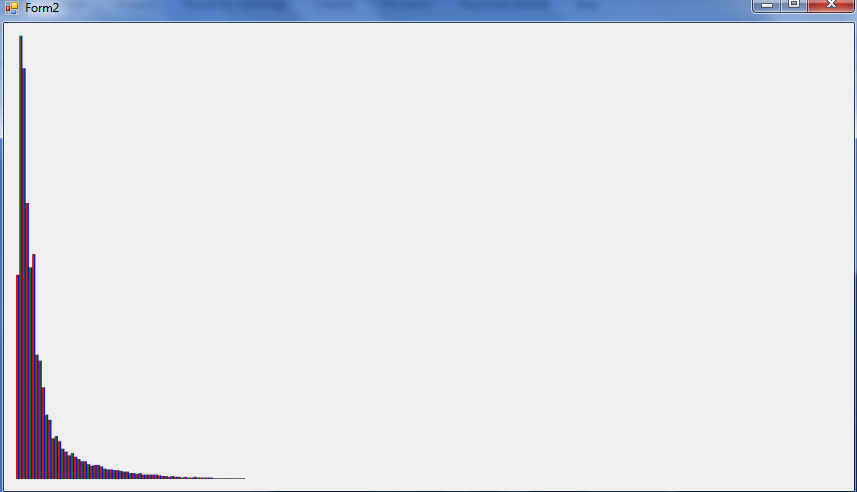


Рисунок 7 – гистограмма высококачественной фильтрации на основе оператора Робертса

**Листинг программы**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

class Calculate

{

public static Image CalculateBarChart(Image image)

{

Bitmap barChart = null;

if (image != null)

{

int width = 768, height = 600;

Bitmap bmp = new Bitmap(image);

Color color;

barChart = new Bitmap(width, height);

int[] R = new int[256];

int[] G = new int[256];

int[] B = new int[256];

int i, j;

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

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

{

color = bmp.GetPixel(i, j);

++R[color.R];

++G[color.G];

++B[color.B];

}

int max = 0;

for (i = 0; i < 256; ++i)

{

if (R[i] > max)

max = R[i];

if (G[i] > max)

max = G[i];

if (B[i] > max)

max = B[i];

}

double point = (double)max / height;

for (i = 0; i < width - 3; ++i)

{

for (j = height - 1; j > height - R[i / 3] / point; --j)

{

barChart.SetPixel(i, j, Color.Red);

}

++i;

for (j = height - 1; j > height - G[i / 3] / point; --j)

{

barChart.SetPixel(i, j, Color.Green);

}

++i;

for (j = height - 1; j > height - B[i / 3] / point; --j)

{

barChart.SetPixel(i, j, Color.Blue);

}

}

}

else

barChart = new Bitmap(1, 1);

return barChart;

}

public static Image lowFrequlencyFilter(Image image)

{

Bitmap picture = new Bitmap(image);

Bitmap picture1 = new Bitmap(image);

Color color;

int r;

//-1 не вошедшее заполняем (их не учитываем при вычислении)

int firstI, firstJ, min = 0;

int[,] filter = new int[5, 5];

for (int i = 0; i < picture.Width; i++)

{

for (int j = 0; j < picture.Height; j++)

{

firstI = i - 2;

color = picture.GetPixel(i, j);

r = color.R;

for (int c = 0; c < 5; c++)

{

firstJ = j - 2;

firstI += c;

for (int p = 0; p < 5; p++)

{

firstJ += p;

if (firstI < 0 || firstI > picture.Width - 1 || firstJ < 0 || firstJ > picture.Height - 1)

{

filter[c, p] = -1;

}

else

{

filter[c, p] = picture.GetPixel(firstI, firstJ).R;

}

}

}

for (int c = 0; c < 5; c++)

{

for (int p = 0; p < 5; p++)

{

if (filter[c, p] != -1)

{

min = filter[c, p];

}

}

}

for (int c = 0; c < 5; c++)

{

for (int p = 0; p < 5; p++)

{

if (filter[c, p] != -1 && filter[c, p] < min)

{

min = filter[c, p];

}

}

}

picture1.SetPixel(i, j, Color.FromArgb(min,min,min));

}

}

return picture1;

}

public static Image highFrequlencyFilter(Image image)

{

Bitmap picture = new Bitmap(image);

Image img = robertsOperator(image);

Bitmap bmp = new Bitmap(img);

Bitmap result = new Bitmap(image);

int r;

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

{

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

{

r = picture.GetPixel(i, j).R;

r += bmp.GetPixel(i, j).R;

if (r > 255)

r = 255;

if (r < 0)

r = 0;

result.SetPixel(i,j,Color.FromArgb(r,r,r));

}

}

return result;

}

public static Image robertsOperator(Image image)

{

int tmp1, tmp2, result, r, r1, r2, r3;

Bitmap picture = new Bitmap(image);

for (int i = 0; i < picture.Width; i++)

{

for (int j = 0; j < picture.Height; j++)

{

r = picture.GetPixel(i, j).R;

if ((i + 1) >= 0 && (i + 1) <= picture.Width-1 && (j + 1) >= 0 && (j + 1) <= picture.Height-1)

{

r1 = picture.GetPixel(i + 1, j + 1).R;

}

else

{

r1 = 255;

}

tmp1 = r - r1;

if ((i + 1) >= 0 && (i + 1) <= picture.Width-1)

{

r2 = picture.GetPixel(i + 1, j).R;

}

else

{

r2 = 0;

}

if ((j + 1) >= 0 && (j + 1) <= picture.Height-1)

{

r3 = picture.GetPixel(i, j + 1).R;

}

else

{

r3 = 0;

}

tmp2 = r2 - r3;

result = (int)Math.Sqrt(Math.Pow(tmp1, 2) + Math.Pow(tmp2, 2));

if (result > 255)

{

result = 255;

}

picture.SetPixel(i, j, Color.FromArgb(result, result, result));

}

}

return picture;

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab3

{

public partial class Form1 : Form

{

private Bitmap inputImage;

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void button\_Open\_Click(object sender, EventArgs e)

{

{

openFileDialog1.Filter = "\*.bmp|\*.jpg|\*.jpeg|";

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

inputImage = new Bitmap(openFileDialog1.FileName);

if (inputImage.Size.Height > 256 && inputImage.Size.Width > 256)

{

pictureBoxOutput.SizeMode = PictureBoxSizeMode.StretchImage;

pictureBoxInput.SizeMode = PictureBoxSizeMode.StretchImage;

pictureBoxInput.Image = Image.FromFile(openFileDialog1.FileName);

}

else

{

inputImage.Dispose();

MessageBox.Show("Wrong inputImage size! Upload another one, please");

}

}

}

}

private void button\_Gistogram\_Click(object sender, EventArgs e)

{

try

{

Form2 f2;

if (pictureBoxOutput.Image != null)

{

f2 = new Form2(Calculate.CalculateBarChart(pictureBoxOutput.Image));

f2.Show();

}

else

{

if (inputImage != null)

{

f2 = new Form2(Calculate.CalculateBarChart(inputImage));

f2.Show();

}

else

throw new ArgumentNullException("All images are empty");

}

}

catch (Exception ex) { MessageBox.Show(ex.ToString()); }

}

private void button\_Clear\_Click(object sender, EventArgs e)

{

try

{

inputImage.Dispose();

pictureBoxInput.Image = null;

pictureBoxOutput.Image = null;

}

catch (Exception) { MessageBox.Show("All images are already empty"); }

}

private void button1\_Click(object sender, EventArgs e)

{

pictureBoxOutput.Image = Calculate.lowFrequlencyFilter(inputImage);

}

private void button2\_Click(object sender, EventArgs e)

{

pictureBoxOutput.Image = Calculate.robertsOperator(inputImage);

}

private void button3\_Click\_1(object sender, EventArgs e)

{

pictureBoxOutput.Image = Calculate.highFrequlencyFilter(inputImage);

}

}

}

**Вывод**: в процессе выполнения лабораторной работы были изучены возможности применения методов пространственной фильтрации изображений для улучшения цифровых изображений.