Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой_____

Преподаватель _____
(подпись)
«_____» _____ 2018 г.

Отчет
по лабораторной работе №_1_

по дисциплине «Интеллектуальные технологии обработки изображений»

ЛР 09.04.04.20.000 О

Студент группы 8ПИ-61 А. Г. Шевелёва
(И. О., Фамилия)

Преподаватель старший преподаватель М.Г. Казаков
должность, ученое звание (И. О., Фамилия)

Барнаул 2018

Постановка задачи:

• Написать основу для представления изображений и их обработки свертками

• Реализовать вычисление частных производныхи оператора Собеля

• Реализовать отображение полученных результатов

Решение:

Свёртка.

Пусть F – входное изображение, H – ядро (2k+1 x 2k+1) (ядро отражено по вертикали и горизонтали), G – выходное изображение

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

Тогда G – результат свертки:

G = H * F

Свёртка обладает следующими свойствами:

- Коммуникативна:
  - $f * g = g * f$
- Ассоциативна:
  - $f * (g * h) = (f * g) * h$
- Дистрибутивна:
  - $f * (g + h) = (f * g) + (f * h)$
- Ассоциативна при умножении на скаляр:
  - $\alpha(f * g) = (\alpha f) * g$

Оператор Собеля.

Оператор Собеля используется для вычисления величины и направления градиента в каждой точке, имеет следующие ядра:

• для вычисления частных производных по X (A — исходное изображение):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A$$

• для вычисления частных производных по Y (A — исходное изображение):

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

Вычисление величины градиента:

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

Вычисление направления градиента:

$$\theta = atan2(G_y, G_x)$$

Результат работы:

Рисунок 1 — Исходное изображение

Рисунок 2 — Вычисление градиента

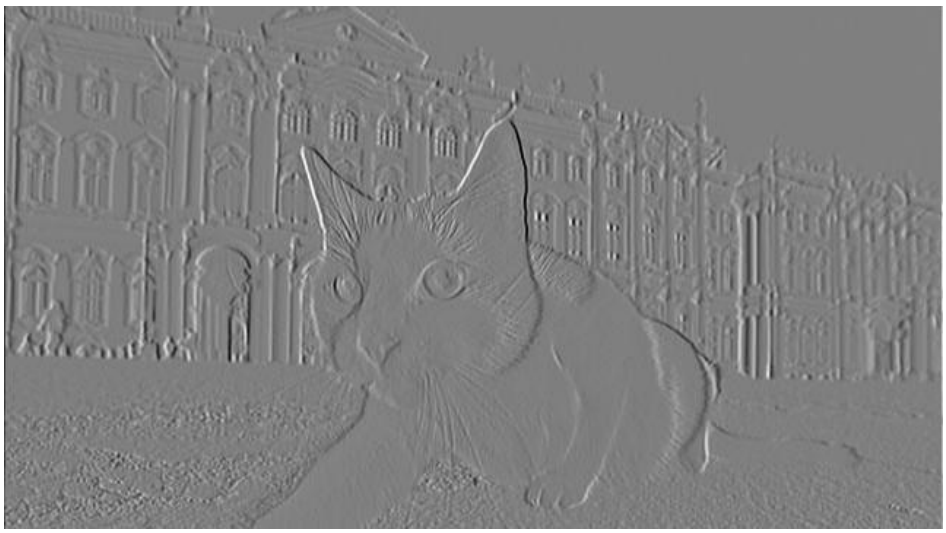

Рисунок 3 — Собель по X

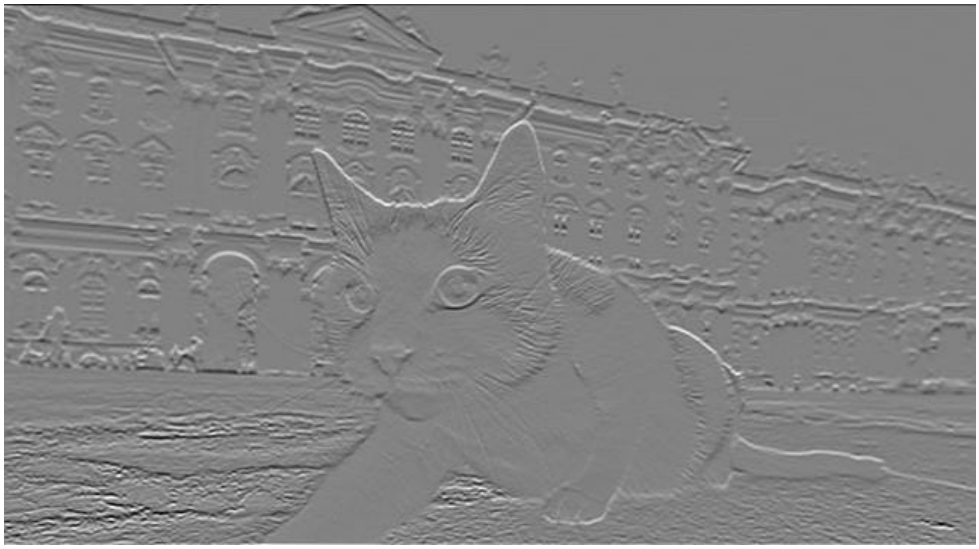

Рисунок 4 — Собель по Y

Исходный код программы:

Image.h

```
enum class Border
{
    Black,
    Copy,
    Reflection,
    Turn
};

class Image
{
private:
    unique_ptr<double []> matrix;
    int width;
    int height;
public:
    Image(int width, int height);
    Image(const Image &image);
    static Image *getSobelXY(Image *imgX, Image *imgY);
```

```
    int getWidth();
    int getHeight();
    QPixmap getImage();
    void changeImage(QPixmap pixmap);
    void saveImage(QString fileName);
    void setPixel(int coordX, int coordY, double value);
    Image *modificateImage(ImageModification *imgModif, Border border);
    void imageMatrixRationing();
    double getImagePixel(int coordX, int coordY, Border border);
    Image* reduceImage(Border border);
private:
    double borderCopyValue(int coordX, int coordY);
    double borderReflectionEdge(int coordX, int coordY);
    double borderTurnImage(int coordX, int coordY);
    double getAverage(int x, int y, Border border);
};
```

ImageModification.h

```
enum class Modification
{
    SobelX,
    SobelY,
    GaussX,
    GaussY
};

class ImageModification
{
private:
    int width;
    int height;
    unique_ptr<double []> matrixModification;
public:
    ImageModification(Modification modification);
    ImageModification(Modification modification, double sigma);
    int getHeight();
    int getWidth();
    double geValuetPixelModification(int coordX, int coordY);
private:
    void feelMatrixSobelX();
    void feelMatrixSobelY();
    int getDimentionRadius(double sigma);
    void feelGaussOneCoord(double sigma);

};
```

Image.cpp

```
Image::Image(int width, int height)
{
    this->width = width;
    this->height = height;
    this->matrix = make_unique<double []>(this->width * this->height);
}

int Image::getHeight(){
    return height;
}

int Image::getWidth(){
    return width;
```

```
}

//замена изображения
void Image::changeImage(QPixmap qPixmap)
{
    QImage qImg = qPixmap.toImage();
    width = qImg.width();
    height = qImg.height();
    matrix.reset();
    matrix = make_unique<double []>(width * height);

    for(int y = 0; y < height; y++)
    {
        for(int x = 0; x < width; x++)
        {
            QColor pixel = (QColor)qImg.pixel(x,y);
            matrix[x + y * width] = (0.213*(pixel.red()) + 0.715*((pixel.green())) + 0.072*((pixel.blue())))/255;
        }
    }
}

QPixmap Image::getImage()
{
    QImage qImg(width, height, QImage::Format_RGB32);
    QColor qColor;
    for(int y = 0; y < height; y++)
        for(int x = 0; x < width; x++)
        {
            qColor.setRgb(matrix[x + y * width]*255, matrix[x + y * width]*255, matrix[x + y * width]*255);
            qImg.setPixel(x, y, qColor.rgb());
        }

    return  QPixmap::fromImage(qImg);
}

double Image::getImagePixel(int coordX, int coordY, Border border){
    if(coordX > -1 && coordX < width && coordY > -1 && coordY < height){
        return matrix[coordX + coordY * width];
    }

    switch(border){
        case Border::Black :
        return 0;
        case Border::Copy:
        return borderCopyValue(coordX, coordY);
        case Border::Reflection:
        return borderReflectionEdge(coordX, coordY);
        default:
        return borderTurnImage(coordX, coordY); //Border::Turn
    }
}

double Image::borderCopyValue(int coordX, int coordY){
    if(coordX < 0)
        coordX = 0;
    if(coordX >= width)
        coordX = width - 1;
    if(coordY < 0)
        coordY = 0;
    if(coordY >= height)
        coordY = height - 1;
    return matrix[coordX + coordY * width];
}
```

```
double Image::borderReflectionEdge(int coordX, int coordY){
    if(coordX < 0)
        coordX = abs(coordX);
    if(coordX >= width)
        coordX = width - (coordX - width) - 1;
    if(coordY < 0)
        coordY = abs(coordY);
    if(coordY >= height)
        coordY = height - (coordY - height) - 1;
    return matrix[coordX + coordY * width];
}

double Image::borderTurnImage(int coordX, int coordY){
    if(coordX < 0)
        coordX = width + coordX;
    if(coordX >= width)
        coordX = 1 + (coordX - width);
    if(coordY < 0)
        coordY = height + coordY;
    if(coordY >= height)
        coordY = 1 + (coordY - height);
    return matrix[coordX + coordY * width];
}

void Image::saveImage(QString fileName){
    QFile file(fileName + ".jpg");
    file.open(QIODevice::WriteOnly);
    getImage().save(&file, "JPG");
}

void Image::setPixel(int coordX, int coordY, double value){
    matrix[coordX + coordY * width] = value;
}

Image* Image::modificateImage(ImageModification *imgModif, Border border){
    Image *resultImage = new Image(this->width, this->height);

    int modHeight = imgModif->getHeight();
    int modWidth = imgModif->getWidth();

    for(int y = 0; y < this->height; y++)
        for(int x = 0; x< this->width; x++){
            double result = 0;
            for(int modY = 0; modY < modHeight; modY++)
                for(int modX = 0; modX < modWidth; modX++){
                    result += imgModif->geValuetPixelModification(modX, modY) *
                            getImagePixel(modWidth/2 + x - modX, modHeight/2 + y - modY, border);
                }
            resultImage->setPixel(x, y, result);
        }
    return resultImage;
}

Image* Image::getSobelXY(Image *imgX, Image *imgY){ //Градиент
    Image *imageXY = new Image(imgX->getWidth(), imgY->getHeight());

    for(int x = 0; x < imageXY->width; x++)
        for(int y = 0; y < imageXY->height; y++){
            imageXY->setPixel(x, y, sqrt(pow(imgX->getImagePixel(x, y, Border::Black),2) +
                                         pow(imgY->getImagePixel(x, y, Border::Black),2)));
        }
    return imageXY;
```

```
}

void Image::imageMatrixRationing(){ //Нормирование
    double max = *std::max_element(&matrix[0], &matrix[width * height]);
    double min = *std::min_element(&matrix[0], &matrix[width * height]);
    double diff = max - min;
    if(diff > 0){
        for(int x = 0; x < width; x++)
            for(int y = 0; y < height; y++){
                this->setPixel(x, y, (this->getImagePixel(x, y, Border::Black) - min)/diff);
            }
    }
}

Image* Image::reduceImage(Border border){
    int newWidth = width/2;
    int newHeight = height/2;
    Image *reduceImage = new Image(newWidth, newHeight);

    for(int x = 0; x < newWidth; x++)
        for(int y = 0; y < newHeight; y++)
        {
            reduceImage->setPixel(x, y, getAverage(x, y, border));
        }

    return reduceImage;
}

double Image::getAverage(int x, int y, Border border){
    x *= 2;
    y *= 2;
    return (getImagePixel(x, y, border) + getImagePixel(x, y + 1, border) +
            getImagePixel(x + 1, y, border) + getImagePixel(x + 1, y + 1, border))/4;
}
```

ImageModification.cpp

```
ImageModification::ImageModification(Modification modification)
{
    switch(modification){
    case Modification::SobelX:
        height = 3;
        width = 3;
        feelMatrixSobelX();
        break;
    case Modification::SobelY:
        height = 3;
        width = 3;
        feelMatrixSobelY();
        break;
    default: ;
    }
}

ImageModification::ImageModification(Modification modification, double sigma)
{
    switch(modification){
    case Modification::GaussX:
        height = 1;
        width = getDimentionRadius(sigma);
        feelGaussOneCoord(sigma);
```

```
        break;
    case Modification::GaussY:
        height = getDimentionRadius(sigma);
        width = 1;
        feelGaussOneCoord(sigma);
        break;
    default: ;
    }
}

void ImageModification::feelMatrixSobelX(){ // { -1, 0, 1, -2, 0, 2, -1, 0, 1 }
    matrixModification =  make_unique<double []>(height * width);
    matrixModification[0] = -1;
    matrixModification[1] = 0;
    matrixModification[2] = 1;
    matrixModification[3] = -2;
    matrixModification[4] = 0;
    matrixModification[5] = 2;
    matrixModification[6] = -1;
    matrixModification[7] = 0;
    matrixModification[8] = 1;
}

void ImageModification::feelMatrixSobelY(){ // { -1, -2, -1, 0, 0, 0, 1, 2, 1 }
    matrixModification =  make_unique<double []>(height * width);
    matrixModification[0] = -1;
    matrixModification[1] = -2;
    matrixModification[2] = -1;
    matrixModification[3] = 0;
    matrixModification[4] = 0;
    matrixModification[5] = 0;
    matrixModification[6] = 1;
    matrixModification[7] = 2;
    matrixModification[8] = 1;
}

int ImageModification::getHeight(){
    return height;
}

int ImageModification::getWidth(){
    return width;
}

double ImageModification::geValuetPixelModification(int coordX, int coordY){
    return matrixModification[coordX + coordY * width];
}

int ImageModification::getDimentionRadius(double sigma){
    int first = (int)(sigma * 3);
    return first * 2;
}

void ImageModification::feelGaussOneCoord(double sigma){
    int dimention = getDimentionRadius(sigma);
    double  multiplier = 1.0/(sqrt(2 * M_PI) * sigma);
    matrixModification = make_unique<double []>(dimention);

    double sum = 0;
    for(int i = 0; i < dimention; i++){
        int coord = i - dimention/2;
        double expon = pow(M_E, ((-1.0) * pow(coord, 2))/(2 * pow(sigma, 2)));
        matrixModification[i] = multiplier * expon;
```

```
        sum += matrixModification[i];
    }

    for(int i = 0; i < dimention; i++){
        matrixModification[i]/=sum;
    }
}
```

Main.cpp

```
void MainWindow::on_pushButton_Sobel_clicked()
{
    //Собель
    ImageModification *imageModifSobX = new ImageModification(Modification::SobelX);
    Image *sobelXImg = image->modificateImage(imageModifSobX, Border::Black);

    ImageModification *imageModifSobY = new ImageModification(Modification::SobelY);
    Image *sobelYImg = image->modificateImage(imageModifSobY, Border::Black);

    Image *sobelXYImg = Image::getSobelXY(sobelXImg, sobelYImg);
    sobelXYImg->imageMatrixRationing();
    showImageAtSecondScreen(sobelXYImg->getImage());
    sobelXYImg->saveImage(filePath + "sobelXYImg");
    //Сохранение Собеля по х и по у
    sobelXImg->imageMatrixRationing();
    sobelXImg->saveImage(filePath + "sobelXImg");
    sobelYImg->imageMatrixRationing();
    sobelYImg->saveImage(filePath + "sobelYImg");
}
```