УДК 004.925

**Разработка программного комплекса обработки изображений для осуществления различных операций над изображениями**

Черников В.А., Старков В.Ю.

ФГБОУ ВО "МГТУ "СТАНКИН" (г. Москва)

*Научный руководитель Илюхин Юрий Владимирович, профессор ФГБОУ ВО "МГТУ "СТАНКИН"*

Аннотация: в статье авторы описывают принципы обработки изображений с помощью различных алгоритмов преобразования и разрабатывают соответствующую программу на языке python

Ключевые слова: обработка изображений, python, qt

# Введение

В данной статье описана программная реализация комплекса обработки осуществления различных операций над изображениями (ПКОИ). К таким операциям относится: построение гистограммы, вычисление полутонового, разностного изображения, умножение изображения на число, сумма двух изображений, сдвиг изображения, обработка операторами Превитта, Собеля и Лапласа. Именно эти операции будут описаны в данной статье.

Данный программный комплекс использует библиотеку Qt для построения графического интерфейса, а алгоритмы обработки изображений написаны с помощью встроенных в Python средств.

В конце статьи будет показаны результаты создания ПКОИ и сделаны соответствующие выводы.

# Основные алгоритмы обработки изображения

В данной главе описаны основные алгоритмы обработки изображения, а также составлены блок-схемы и представлена программная реализация описываемого алгоритма.

## 2.1 Вычисление гистограммы яркостей



**Рис. 1**. Блок-схема гистограммы яркостей

Алгоритм имеет сложность по времени O(W∗H)O(W\*H)O(W∗H),
где W - количество пикселей в ширину, а H количество пикселей в высоту.

Программная реализация показана ниже:

def make\_mult(img1: QImage, Mult\_SpinBox: QDoubleSpinBox):

 for x in range(img1.width()):

 for y in range(img1.height()):

 color\_1 = img1.pixelColor(x, y)

 d\_r = color\_1.red()\*Mult\_SpinBox

 if d\_r > 255:

 d\_r = 255

 d\_g = color\_1.green()\*Mult\_SpinBox

 print(Mult\_SpinBox)

 if d\_g > 255:

 d\_g = 255

 d\_b = color\_1.blue()\*Mult\_SpinBox

 if d\_b > 255:

 d\_b = 255

 new\_color = QColor(d\_r, d\_g, d\_b)

 img1.setPixelColor(x, y, new\_color)

 return img1

## 2.2 Сдвиг изображения



**Рис. 2**. Блок-схема сдвига изображения

Алгоритм имеет сложность по времени O(W∗H)O(W\*H)O(W∗H),
где W - количество пикселей в ширину, а H количество пикселей в высоту.

Программная реализация показана ниже:

def make\_move(img1: QImage, dX:QSpinBox, dY:QSpinBox):

 img=QImage(img1)

 for x in range(1,img1.height()-1):

 for y in range(1,img1.width()-1):

 img.setPixelColor(y,x,QColor(0,0,0))

 if (dY >=0): #Смещение вниз

 for x in range(1,img1.height()-dY):

 if (dX>=0): #Смещение вниз/вправо

 for y in range(1,img1.width()-dX):

 img.setPixelColor(y+dX, x+dY, img1.pixelColor(y,x))

 else: #вниз/влево

 for y in range(1,img1.width()+dX):

 img.setPixelColor(y, x+dY, img1.pixelColor(y-dX,x))

 else: #смещение вверх

 for x in range(1,img1.height()+dY):

 if (dX>=0): #Смещение вверх/вправо

 for y in range(1,img1.width()-dX):

 img.setPixelColor(y+dX, x, img1.pixelColor(y,x-dY))

 else: #Смещение вверх/влево

 for y in range(1,img1.width()+dX):

 img.setPixelColor(y,x,img1.pixelColor(y-dX,x-dY))

 return img

## 2.3 Умножение изображения на число

**Рис. 3.** Блок-схема умножения изображения на число

Алгоритм имеет сложность по времени O(3∗W∗H)O(3\*W\*H)O(3∗W∗H),
где W - количество пикселей в ширину, а H количество пикселей в высоту.

Программная реализация показана ниже:

def make\_mult(img1: QImage, Mult\_SpinBox: QDoubleSpinBox):

 for x in range(img1.width()):

 for y in range(img1.height()):

 color\_1 = img1.pixelColor(x, y)

 d\_r = color\_1.red()\*Mult\_SpinBox

 if d\_r > 255:

 d\_r = 255

 d\_g = color\_1.green()\*Mult\_SpinBox

 print(Mult\_SpinBox)

 if d\_g > 255:

 d\_g = 255

 d\_b = color\_1.blue()\*Mult\_SpinBox

 if d\_b > 255:

 d\_b = 255

 new\_color = QColor(d\_r, d\_g, d\_b)

 img1.setPixelColor(x, y, new\_color)

 return img1

## 2.4 Бинаризация изображения по заданному порогу

**Рис. 4.** Блок-схема бинаризации по заданному порогу

Алгоритм имеет сложность по времени O(3∗W∗H)O(3\*W\*H)O(3∗W∗H),
где W - количество пикселей в ширину, а H количество пикселей в высоту.

Программная реализация показана ниже:

def make\_bin (img1: QImage, threshold\_SpinBox: QDoubleSpinBox):

 for x in range(img1.width()):

 for y in range(img1.height()):

 color\_1 = img1.pixelColor(x, y)

 d\_r = color\_1.red()

 if d\_r > threshold\_SpinBox:

 d\_r = 255

 else :

 d\_r = 0

 d\_g = color\_1.green()

 if d\_g > threshold\_SpinBox:

 d\_g = 255

 else :

 d\_g = 0

 d\_b = color\_1.blue()

 if d\_b > threshold\_SpinBox:

 d\_b = 255

 else :

 d\_b = 0

 new\_color = QColor(d\_r, d\_g, d\_b)

 img1.setPixelColor(x, y, new\_color)

 return img1

## 2.5 Оператор Лапласа

**Рис. 5.** Блок-схема оператора Лапласа

Алгоритм имеет оценку сложности по времени O((9+1)∗W∗H+W∗H)O((9+1)\*W\*H+W\*H)O((9+1)∗W∗H+W∗H),
где W - количество пикселей в ширину, а H количество пикселей в высоту. 9 - количество элементов в операторе Лапласа.

Программная реализация показана ниже:

def make\_laplas(img1: QImage):

 img=QImage(img1)

 mask=[[0,-1,0],

 [-1,4,-1],

 [0,-1,0]]

 R=0

 G=0

 B=0

 H=img1.height()

 W=img1.width()

 matrix=numpy.zeros([H,W])

 max\_color\_value=0

 for x in range(1,img1.height()-1):

 for y in range(1,img1.width()-1):

 new\_color\_value=0

 new\_y=0

 for hw in [-1,0,1]:

 for wi in [-1,0,1]:

 new\_color = int((img1.pixelColor(y+wi, x+hw).red()+ img1.pixelColor(y+wi, x+hw).green()+img1.pixelColor(y+wi, x+hw).blue())/ 3)

 new\_color\_value+=mask[hw+1][wi+1]\*new\_color

 if (math.fabs(new\_color\_value)>max\_color\_value):

 max\_color\_value=math.fabs(new\_color\_value)

 matrix[x][y]=new\_color\_value

 for x in range(1,img1.height()-1):

 for y in range(1,img1.width()-1):

 R=matrix[x][y]/max\_color\_value\*127+128

 G=matrix[x][y]/max\_color\_value\*127+128

 B=matrix[x][y]/max\_color\_value\*127+128

 qColor=QColor(R,G,B)

 img.setPixelColor(y,x,qColor)

 return img

## 2.6 Оператор Превитта

**Рис. 6.** Блок-схема оператора Превитта

Алгоритм имеет оценку сложности по времени O(W∗H∗(9+2))O(W\*H\*(9+2))O(W∗H∗(9+2)),
где W - количество пикселей в ширину, а H количество пикселей в высоту,
9 - количество элементов в операторе Превитта,
2 - количество условий на одну итерацию.

Программная реализация показана ниже:

def make\_prewitt (img1: QImage):

 img=QImage(img1)

 Hx=[[1,1,1],

 [0,0,0],

 [-1,-1,-1]]

 Hy=[[-1,0,1],

 [-1,0,1],

 [-1,0,1]]

 for x in range(1,img1.height()-1):

 for y in range(1,img1.width()-1):

 new\_x=0

 new\_y=0

 for hw in [-1,0,1]:

 for wi in [-1,0,1]:

 new\_color = int((img1.pixelColor(y+wi, x+hw).red()+ img1.pixelColor(y+wi, x+hw).green()+img1.pixelColor(y+wi, x+hw).blue())/ 3)

 new\_x+=Hx[hw+1][wi+1]\*new\_color

 new\_y+=Hy[hw+1][wi+1]\*new\_color

 new\_color1=int(math.sqrt(new\_x\*\*2+new\_y\*\*2))

 if new\_color1>255:

 new\_color1=255

 if new\_color1<0:

 new\_color1=0

 qColor = QColor(new\_color1, new\_color1, new\_color1)

 img.setPixelColor(y, x, qColor)

 return img

## 2.7 Оператор Собеля



**Рис. 7.** Блок-схема оператора Собеля

Алгоритм имеет оценку сложности по времени O(W∗H∗(9+2))O(W\*H\*(9+2))O(W∗H∗(9+2)),
где W - количество пикселей в ширину, а H количество пикселей в высоту,
9 - количество элементов в операторе Собеля,
2 - количество условий на одну итерацию.

Программная реализация показана ниже:

def make\_sobel (img1: QImage):

 img=QImage(img1)

 Hx=[[-1,0,1],

 [-2,0,2],

 [-1,0,1]]

 Hy=[[-1,-2,-1],

 [0,0,0],

 [1,2,1]]

 for x in range(1,img1.height()-1):

 for y in range(1,img1.width()-1):

 new\_x=0

 new\_y=0

 for hw in [-1,0,1]:

 for wi in [-1,0,1]:

 new\_color = int((img1.pixelColor(y+wi, x+hw).red()+ img1.pixelColor(y+wi, x+hw).green()+img1.pixelColor(y+wi, x+hw).blue())/ 3)

 new\_x+=Hx[hw+1][wi+1]\*new\_color

 new\_y+=Hy[hw+1][wi+1]\*new\_color

 new\_color1=int(math.sqrt(new\_x\*\*2+new\_y\*\*2))

 if new\_color1>255:

 new\_color1=255

 if new\_color1<0:

 new\_color1=0

 qColor = QColor(new\_color1, new\_color1, new\_color1)

 img.setPixelColor(y, x, qColor)

 return img

# Результат проделанной работы

По итогу проделанной работы, была получена работоспособная программа, в которой можно осуществлять все выше перечисленные операции. Интерфейс программы представляет собой 1 окно, состоящее из 4х позиций для работы с изображениями (рис.8). Первая позиция необходимы для демонстрации исходного изображения. В нем также, как и в других можно построить гистограмму. Для того чтобы сложить или дифференцировать изображение, его необходимо переместить в одно и 3х оставшихся окошек. Если операция требует для работы 2 изображения, то в этой команде нужно выбрать из каких окон будут взяты картинки.

**Рис. 8.** Интерфейс программы

## 3.1 Примеры работы программы

Построение гистограммы приставлено на рисунке 9, для примера взято изображение книги.

Рис. 9. Пример построения гистограммы

На рисунке видно какое количество пикселей имеет тот или иной цвет. Благодаря данной гистограмме, пользователь имеет возможность бинаризировать изображение по любому порогу. Например, на рисунке 10 представлен порог бинаризации 200.

Рис. 10. Бинаризация изображения

Также с помощью данной программы можно получить полутоновое изображение, пример представлен на рисунке 11.

Рис. 11. Полутоновое изображение

# Заключение

Итак, был разработан программный комплекс обработки изображений. Был описан принцип действия работы каждого алгоритма на блок-схемах и разработана соответствующая программная составляющая. Благодаря дружелюбному интерфейсу и простоте программы можно быстро и удобно обрабатывать изображение с помощью гистограммы яркостей, сдвигать изображение, умножать его на число, бинаризировать по заданному порогу и использовать операторы Лапласа, Превитта и Собеля.