МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра механики, мехатроники и робототехники

УТВЕРЖДАЮ
проректор по учебной работе
_____ О.Г. Локтионова
«15» 12 2017 г.

МЕТОДЫ ПОДЧЁРКИВАНИЯ КОНТУРОВ НА ИЗОБРАЖЕНИЯХ В СИСТЕМАХ ТЕХНИЧЕСКОГО ЗРЕНИЯ МОБИЛЬНЫХ РОБОТОВ С ИСПОЛЬЗОВАНИЕМ ПРОГРАММНЫХ ПАКЕТОВ

Методические указания к выполнению лабораторной работы
по дисциплине «Информационные системы роботов и обработка
сигналов» для студентов направления
15.04.06 Мехатроника и робототехника

Курск

УДК 621

Составители: А.В. Мальчиков, С.И. Савин

Рецензент
Кандидат технических наук, доцент *Е.Н. Политов*

Методы подчёркивания контуров на изображениях в системах технического зрения мобильных роботов с использованием программных пакетов: методические указания к выполнению лабораторной работы по дисциплине «Информационные системы роботов и обработка сигналов» / Юго-Зап. гос. ун-т; сост. А.В. Мальчиков, С.И. Савин. Курск, 2017. 22 с.: ил. 8, табл. 1. Библиогр.: с. 21-22.

Методические указания содержат сведения о методах обработки изображений, в частности, методах подчёркивания контуров путём использования градиента и операторов Робертса, Собела и Лапласа, а также описание и порядок выполнения лабораторной работы.

Методические указания соответствуют требованиям программы, утверждённой учебно-методическим объединением (УМО).

Предназначены для студентов специальности 15.04.06 всех форм обучения.

Текст печатается в авторской редакции

Подписано в печать 15.12.17 . Формат 60х84 1/16
Усл.печ.л. 1,3. Уч.-изд.л. 1,2 Тираж 10 экз. Заказ 353 Бесплатно.
Юго-Западный государственный университет.
305040 Курск, ул. 50 лет Октября, 94

МЕТОДЫ ПОДЧЁРКИВАНИЯ КОНТУРОВ НА ИЗОБРАЖЕНИЯХ В СИСТЕМАХ ТЕХНИЧЕСКОГО ЗРЕНИЯ МОБИЛЬНЫХ РОБОТОВ С ИСПОЛЬЗОВАНИЕМ ПРОГРАММНЫХ ПАКЕТОВ

Цель работы: *изучение приёмов разработки программ для подчёркивания контуров на изображениях путем использования оператора Робертса, Собела, Лапласа и градиента.*

Объект исследования: *оператор Робертса, оператор Собела, оператор Лапласа, градиент.*

Аппаратные средства: *программа «виртуальный лабораторный комплекс Vision Lab», среда программирования C++ Builder.*

Формируемые компетенции:

ПК-2 - способность использовать имеющиеся программные пакеты и, при необходимости, разрабатывать новое программное обеспечение, необходимое для обработки информации и управления в мехатронных и робототехнических системах, а также для их проектирования

1. Краткие теоретические сведения

Во многих случаях наиболее информативными являются характеристики границ некоторых областей изображения – контуров. Контуром можно назвать пространственно протяженный перепад (скачкообразное изменение) значений яркости или цветовой составляющей. Процедура выделения контура может быть сведена к последовательно выполненным процедурам подчеркивания контура и пороговой обработки. Существенными проблемами при выделении контуров являются разрывы контуров в местах с плавно меняющейся яркостью, появление ошибочных контуров (в случае высокой зашумлённости), излишне толстые контурные линии.

1.1 Подчеркивание контуров

Процедура подчеркивания контуров представляет собой своего рода дифференцирование изображения, т.е. яркость (или цветность) элементов, имеющих существенные отличия от

соседних (по яркости или цветности) будет выше, чем у элементов, сходных с соседними. Для представления изображения в виде двумерного массива элементов существует ряд методов подчеркивания контуров. Часто используются нахождение градиента, оператор Робертса, оператор Собела, - их называют градиентными методами, а так же оператор Лапласа.

1.2 Градиент

Данный метод можно считать менее точным, в сравнении с методами Робертса и Собела. Вместе с тем данный метод требует меньшего объёма вычислений. Следует так же отметить, что отличие приближенных значений градиента от точных может достигать $\sqrt{2}$ раз (это проявляется в контурных линиях, наклонённых на 45° к координатным осям).

Математическое представление градиента:

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \qquad (1),$$

где f – массив элементов изображения, x и y – «координаты»[2] изображения.

Выражение (1) может быть записано следующим образом:

$$Y_{i,j} = \sqrt{(X_{i,j} - X_{i+1,j})^2 + (X_{i,j} - X_{i,j+1})^2} \qquad (2),$$

где i и j – индексы элементов изображения, X – исходное изображение, Y – обработанное изображение.

Зачастую, на практике вместо выражения (2) используют приближенное выражение (3):

$$Y_{i,j} = \left|X_{i,j} - X_{i+1,j}\right| + \left|X_{i,j} - X_{i,j+1}\right| \qquad (3).$$

Так же градиент может быть вычислен наложением масок:

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix} \text{ и } \begin{pmatrix} 1 & -1 \end{pmatrix}$$

Чтобы применить реализованный в виртуальном лабораторном комплексе Vision Lab градиент необходимо выбрать вкладку «Подчёркивание контуров» и нажать на кнопку

«Градиент» или «Градиент (X)», для применения градиента со сглаживанием. В программе реализован метод, описываемый выражением (2).

На рисунке 1 представлено изображение до и после обработки градиентом. Следует отметить, что визуальные отличия изображений, обработанных различными градиентными методами незначительны. Исходя из этого, иллюстрации к некоторым другим градиентным методам не приведены.

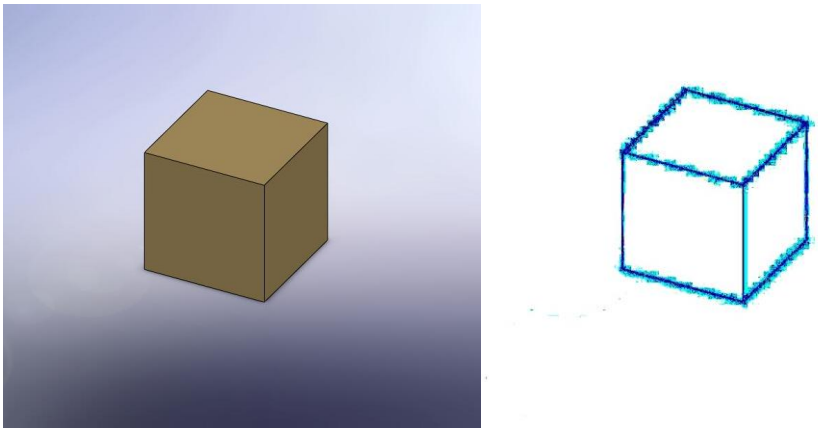

Рис. 1 Изображение до и после обработки градиентом; обработанное изображение показано справа

3

1.3 Оператор Робертса

Оператор Робертса даёт другой простой метод вычисления дискретного градиента. При его построении используется тот факт, что для вычисления модуля градиента можно использовать производные (разности) в двух взаимно перпендикулярных направлениях. В операторе Робертса берутся диагональные разности.

Математическое представление оператора Робертса:

$$Y_{i,j} = \sqrt{(X_{i,j} - X_{i+1,j+1})^2 + (X_{i+1,j} - X_{i,j+1})^2} \qquad (4)$$

Зачастую, на практике вместо выражения (4) используют приближенное выражение (5):

$$Y_{i,j} = \left|X_{i,j} - X_{i+1,j+1}\right| + \left|X_{i+1,j} - X_{i,j+1}\right| \qquad (5)$$

Так же оператор Робертса может быть заменён наложением масок:

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \text{ и } \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

Чтобы применить реализованный в виртуальном лабораторном комплексе Vision Lab оператор Робертса необходимо выбрать вкладку «Подчёркивание контуров» и нажать на кнопку «Оператор Робертса» или «Оператор Робертса (X)», для применения оператора Робертса со сглаживанием. В программе реализован метод, описываемый выражением (4).

1.4 Оператор Собела

Оператор Собела в отличие от стандартного градиента и оператора Робертса, для вычисления которых требуется 3 и 4 элемента изображения соответственно использует 8 элементов изображения. В связи с этим использование оператора Собела создаёт значительно большую нагрузку на вычислитель, в сравнении с оператором Робертса и стандартным градиентом.

Математическое представление оператора Собела:

$$Y_{i,j} = \sqrt{DX_{i,j}{}^2 + DY_{i,j}{}^2} \qquad (6),$$

где

$$DX_{i,j} = \sqrt{\left(X_{i-1,j-1} + 2 \cdot X_{i-1,j} + X_{i-1,j+1}\right)^2 + \left(X_{i+1,j-1} + 2 \cdot X_{i+1,j} + X_{i+1,j+1}\right)^2}$$

$$DY_{i,j} = \sqrt{\left(X_{i-1,j-1} + 2 \cdot X_{i,j-1} + X_{i+1,j-1}\right)^2 + \left(X_{i-1,j+1} + 2 \cdot X_{i,j+1} + X_{i+1,j+1}\right)^2}$$

Зачастую, на практике вместо выражения (6) используют приближенное выражение (7):

$$Y_{i,j} = \left|DX^*{}_{i,j}\right| + \left|DY^*{}_{i,j}\right| \qquad (7),$$

где:

$$DX^*{}_{i,j} = \left|X_{i-1,j-1} + 2 \cdot X_{i-1,j} + X_{i-1,j+1}\right| + \left|X_{i+1,j-1} + 2 \cdot X_{i+1,j} + X_{i+1,j+1}\right|$$

$$DY^*{}_{i,j} = \left|X_{i-1,j-1} + 2 \cdot X_{i,j-1} + X_{i+1,j-1}\right| + \left|X_{i-1,j+1} + 2 \cdot X_{i,j+1} + X_{i+1,j+1}\right|$$

Так же оператор Собела может быть заменён наложением масок:

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \text{ и } \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Чтобы применить реализованный в виртуальном лабораторном комплексе Vision Lab оператор Собела необходимо выбрать вкладку «Подчёркивание контуров» и нажать на кнопку «Оператор Собела» или «Оператор Собела (X)», для применения оператора Собела со сглаживанием. В программе реализован метод, описываемый выражением (6).

1.5 Оператор Лапласа

Для решения задачи выделения перепадов яркости или цвета можно использовать дифференциальные операторы высоких порядков, например оператор Лапласа. Следует отметить, что оператор Лапласа может давать разные по знаку значения, в зависимости от направления перепада (уменьшение или увеличение).

Математическое представление оператора Лапласа:

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{dx^2} + \frac{\partial^2 f(x,y)}{dy^2} \qquad (8).$$

Для дискретной области процедура подчеркивания контуров оператором Лапласа может быть записана следующим образом:

$$Y_{i,j} = DX_{i,j} + DY_{i,j} \qquad (9),$$

где: $DX_{i,j} = X_{i-1,j} - 2 \cdot X_{i,j} + X_{i+1,j}$ и $DY_{i,j} = X_{i,j-1} - 2 \cdot X_{i,j} + X_{i,j+1}$

Оператор Лапласа может быть заменён наложением маски:

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

В виртуальном лабораторном комплексе Vision Lab реализован оператор Лапласа, совмещенный с пороговой обработкой. Чтобы применить этот метод необходимо выбрать вкладку «Подчёркивание контуров» и нажать на кнопку «Оператор

Лапласа». В появившемся диалоговом окне необходимо установить пороговые значения для отрицательных и положительных значений.

На рисунке 2 представлено изображение до и после обработки оператором Лапласа.

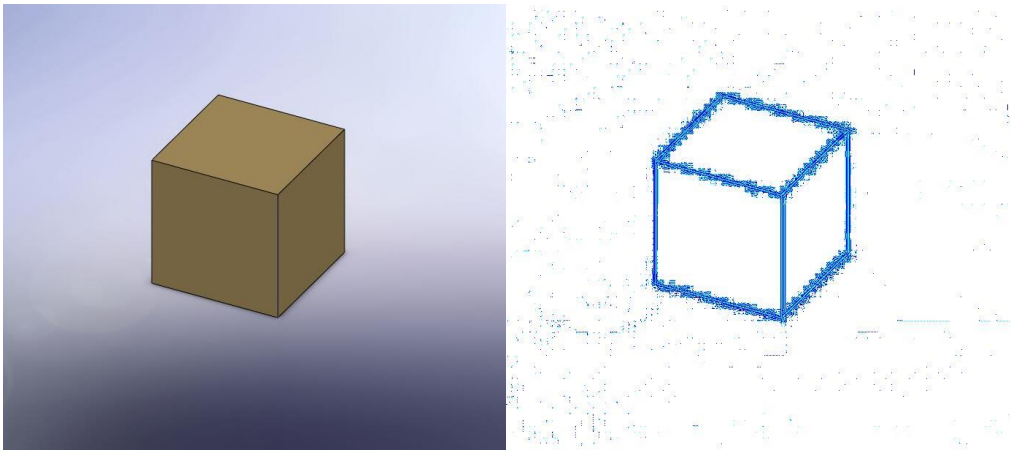

Рис. 2 Изображение до и после обработки оператором Лапласа; исходное изображение слева

2. Методика выполнения лабораторной работы

6

Первая часть лабораторной работы заключается в обработке изображения с использованием виртуального лабораторного комплекса Vision Lab.

Вторая часть лабораторной работы заключается в реализации алгоритмов пороговой обработки на языке C++ в среде программирования C++ Builder.

Для того, чтобы организовать обработку изображения необходимо создать пользовательский интерфейс, позволяющий загружать изображения. Один из вариантов подобного интерфейса показан на рисунке 3.

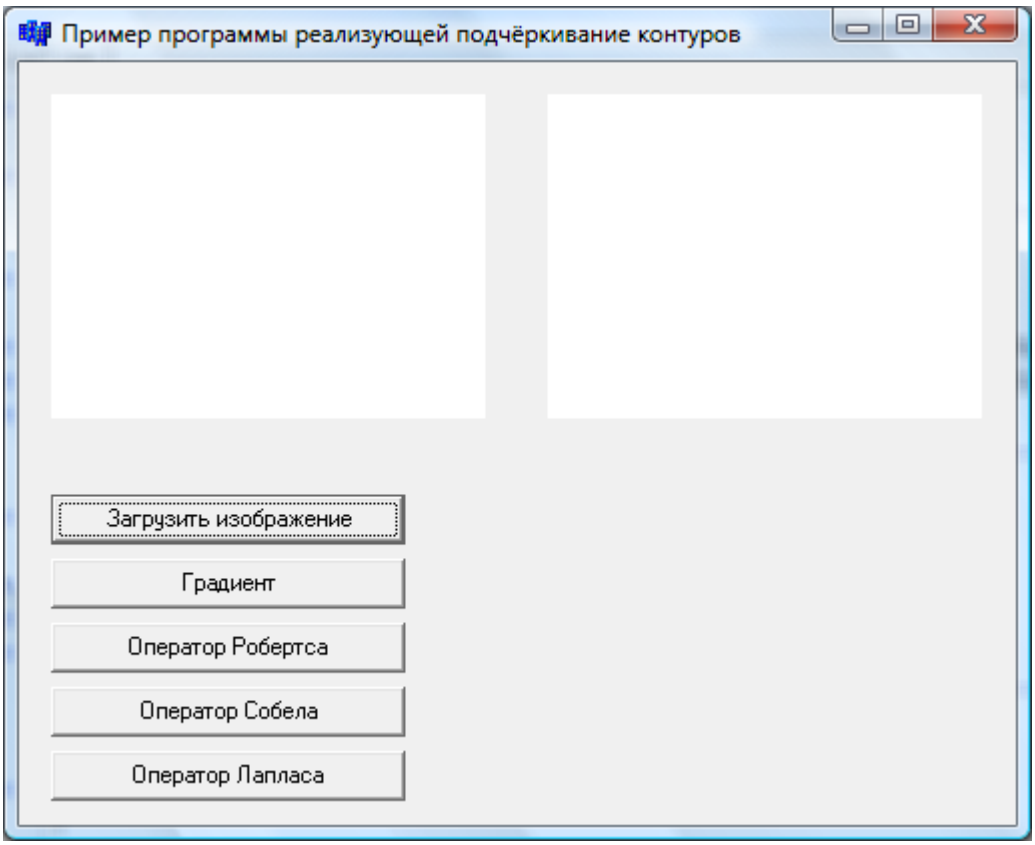

Рис. 3 Пример пользовательского интерфейса

Предложенный пример пользовательского интерфейса, представляет собой четыре компонента из стандартной библиотеки C++ Builder – два компонента Image и пять компонентов Button. Компонент Image расположен на вкладке Additional панели компонентов (см. рисунок 4).

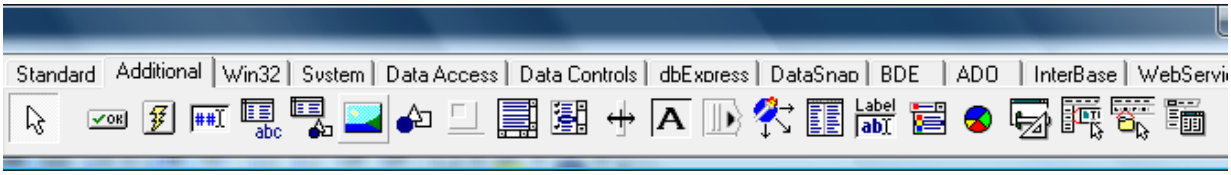

Рис. 4 Вкладка Additional панели компонентов C++ Builder

Компонент Button расположен на вкладке Standard панели компонентов (см. рисунок 5).

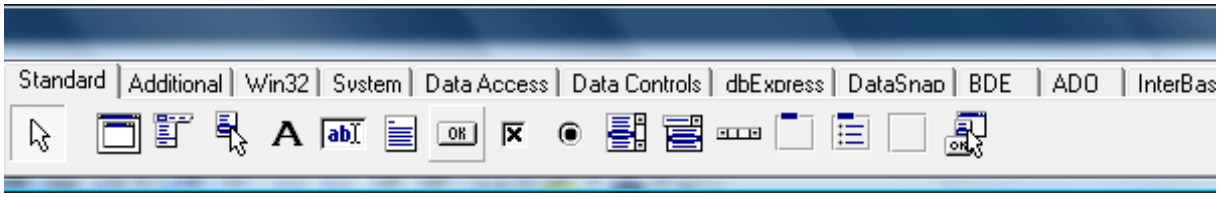

Рис. 5 Вкладка Standard панели компонентов C++ Builder

После размещения компонентов на форме следует настроить их свойства. Это может быть произведено путём изменения их свойств в окне Object Inspector. Окно Object Inspector отображающее свойства компонента Button показано на рисунке 6.

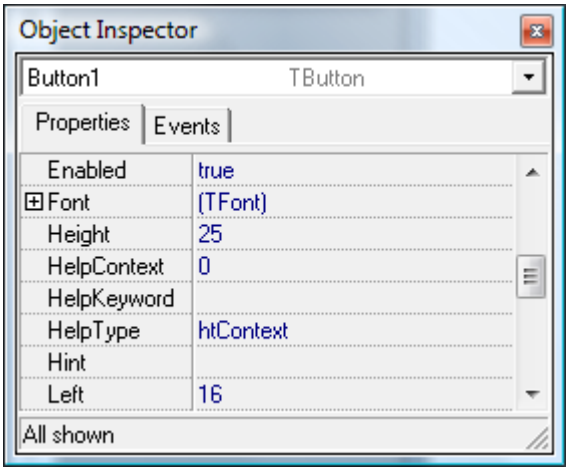

Рис. 6 Окно Object Inspector отображающее свойства компонента Button

В предложенном примере были изменены свойства Name для компонентов Button, свойства Proportional и Stretch компонентов Image и свойства, определяющие положение компонентов на форме[8]. Также, для того, чтобы сделать компоненты Image видимыми пользователю необходимо загрузит в них некоторые изображения. Для этой цели можно использовать изображение, представляющее собой массив точек белого цвета. Можно загрузить такое изображение, используя окно Object Inspector для вызова диалогового окна Picture Editor. Диалоговое окно Picture Editor показано на рисунке 7.

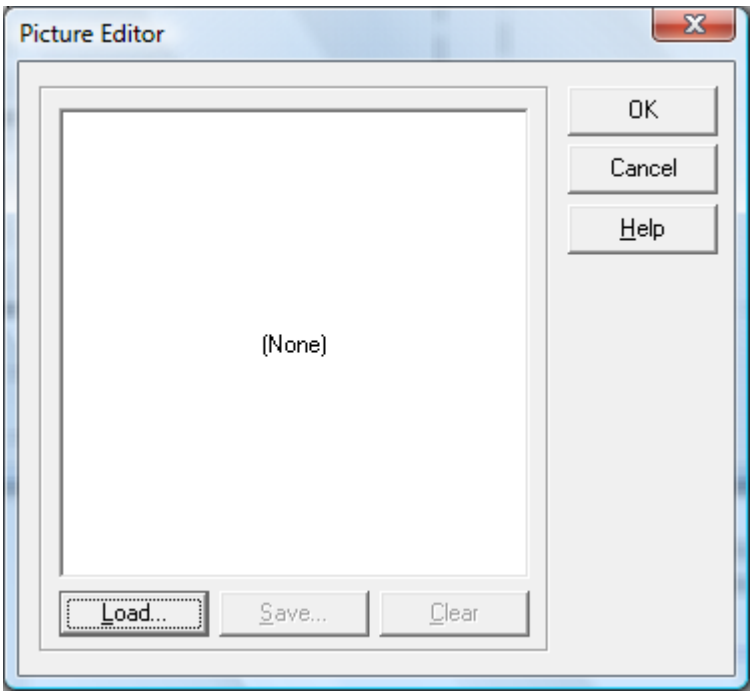

Рис. 7 Диалоговое окно Picture Editor

Также для загрузки изображений, которые будут отображаться по умолчанию можно использовать специальный код (см. листинг 1)

9

Листинг 1 Пример программного кода, реализующего загрузку изображения по умолчанию

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Image1->Picture->LoadFromFile(ExtractFilePath( Application->ExeName ) + "\Начальное изображение.bmp");
Image2->Picture->LoadFromFile(ExtractFilePath( Application->ExeName ) + "\Начальное изображение.bmp");
}
```

Следует отметить, что функцию FormCreate среда C++ Builder может сгенерировать автоматически при двойном щелчке по форме.

После окончания работы над пользовательским интерфейсом следует разработать функции, которые будут выполняться при нажатии на кнопки (компоненты Button). В приведённом примере одна из кнопок выполняет загрузку изображения в один из компонентов Image, вторая выполняет обработку изображения.

Для организации загрузки изображения имеет смысл воспользоваться стандартным компонентом OpenDialog. Компонент OpenDialog расположен на вкладке Dialogs панели компонентов (см. рисунок 8).

Рис. 8 Вкладка Dialogs панели компонентов C++ Builder

Загрузку изображений с использованием компонента OpenDialog реализует код показанный на листинге 2.

Листинг 2 Пример программного кода, реализующего загрузку изображения с использованием компонента OpenDialog

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (OpenDialog1->Execute())
Image1->Picture->LoadFromFile(OpenDialog1->FileName);
}
```

0

Для организации алгоритма подчёркивания контуров на изображении нужно преобразовать формат входных данных. Загруженное изображение представляет собой массив данных типа **TColor**. Нам необходимо преобразовать его, получив на выходе массив данных типа **int**, содержащий данные о яркости точек изображения. Для преобразования переменной типа **TColor** в переменную типа **int**, содержащую значение яркости, можно использовать код, показанный на листинге 3.

Листинг 3 Пример программного кода, реализующего преобразование данных типа TColor в данные типа int содержащие информацию о яркости.

```
int PixelBright(TColor Color)
{
int Blue, Green, Red;
int Result;
```

```
Blue = Color / (256*256);
Green = ( Color - (Blue * (256*256)) ) / 256;
Red = ( Color - (Blue * (256*256)) - (Green * 256) );

Result = Blue + Green + Red;
return Result;
}
```

Используя данную функцию, представляется возможным реализовать алгоритм обработки изображения градиентом. Код, реализующий данный вид обработки, показан на листинге 4.

Листинг 4 Код, реализующий обработку изображения градинтом.

```
void SizeKorrection(TImage *Input, TImage *Output)
{
if ((Input->Picture->Bitmap->Width != Output->Picture->Bitmap->Width) ||
   (Input->Picture->Bitmap->Height != Output->Picture->Bitmap->Height))
Output->Picture = Input->Picture;
}

void Artefact_2x2(TImage *Output)
{
int CountX, CountY;
int i, j;

CountX = Output->Picture->Bitmap->Width;
CountY = Output->Picture->Bitmap->Height;

for (i = 0; i < CountY; i++)
Output->Canvas->Pixels[CountX - 1][i] = Output->Canvas->Pixels[CountX - 2][i];

for (j = 0; j < CountX; j++)
Output->Canvas->Pixels[j][CountY - 1] = Output->Canvas->Pixels[j][CountY - 2];
}
```

```
void Gradient_normal(TImage *Input, TImage *Output)
{
int CountX, CountY;
int i, j;
int A,B,C;
float Calculet;
TColor OutColor;

SizeKorrection(Input, Output);

CountX = Input->Picture->Bitmap->Width;
CountY = Input->Picture->Bitmap->Height;

for (i = 0; i < CountY - 1; i++)
for (j = 0; j < CountX - 1; j++)
{
A = PixelBright(Input->Canvas->Pixels[j][i]);
B = PixelBright(Input->Canvas->Pixels[j + 1][i]);
C = PixelBright(Input->Canvas->Pixels[j][i + 1]);

Calculet = Calculet / 3;
Calculet = sqrt( ((A - B)*(A - B)) + ((A - C)*(A - C)) );


OutColor = (TColor) (Calculet + (Calculet * 256) + (Calculet * 256 *
256));
Output->Canvas->Pixels[j][i] = OutColor;
}

Artefact_2x2(Output);
}
```

2

Приведённый выше программный код реализует обработку всего изображения и загрузку результатов во второй компонент типа TImage. Вызов функции Gradient_normal производится из функции обработки нажатия кнопки на форме. Код, реализующий вызов функции Gradient_normal, показан на листинге 5.

Листинг 5 Пример программного кода, реализующего вызов функции Gradient_normal.

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Gradient_normal(Image1, Image2);
}
```

Код, реализующий обработку изображения оператором Робертса, показан на листинге 6.

Листинг 6 Пример программного кода, реализующего обработку изображения оператором Робертса.

```
void Gradient_Roberts(TImage *Input, TImage *Output)
{
int CountX, CountY;
int i, j;
int A,B,C,D;
float Calculet;
TColor OutColor;

SizeKorrection(Input, Output);

CountX = Input->Picture->Bitmap->Width;
CountY = Input->Picture->Bitmap->Height;

for (i = 0; i < CountY - 1; i++)
for (j = 0; j < CountX - 1; j++)
{
A = PixelBright(Input->Canvas->Pixels[j][i]);
B = PixelBright(Input->Canvas->Pixels[j + 1][i]);
C = PixelBright(Input->Canvas->Pixels[j][i + 1]);
D = PixelBright(Input->Canvas->Pixels[j + 1][i + 1]);

Calculet = sqrt( ((A - D)*(A - D)) + ((B - C)*(B - C)) );
Calculet = Calculet / 3;

OutColor =(TColor) (Calculet + (Calculet * 256) + (Calculet * 256 * 256));
```

```
Output->Canvas->Pixels[j][i] = OutColor;
}

Artefact_2x2(Output);
}
```

Код, реализующий вызов функции Gradient_Roberts, показан на листинге 7.

Листинг 7 Пример программного кода, реализующего вызов функции Gradient_Roberts.

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Gradient_Roberts(Image1, Image2);
}
```

Код, реализующий обработку изображения оператором Собела, показан на листинге 8.

Листинг 8 Пример программного кода, реализующего обработку изображения оператором Собела.

```
void Artefact_3x3(TImage *Output)
{
int CountX, CountY;
int i, j;

CountX = Output->Picture->Bitmap->Width;
CountY = Output->Picture->Bitmap->Height;

for (i = 0; i < CountY; i++)
Output->Canvas->Pixels[CountX - 1][i] = Output->Canvas->Pixels[CountX - 2][i];

for (j = 0; j < CountX; j++)
Output->Canvas->Pixels[j][CountY - 1] = Output->Canvas->Pixels[j][CountY - 2];

for (i = 0; i < CountY; i++)
```

```
Output->Canvas->Pixels[0][i] = Output->Canvas->Pixels[1][i];

for (j = 0; j < CountX; j++)
Output->Canvas->Pixels[j][0] = Output->Canvas->Pixels[j][1];
}

void Gradient_Sobel(TImage *Input, TImage *Output)
{
int CountX, CountY;
int i, j;
int A,B,C,D,E,F,G,H;
float Calc_G,Calc_V,Calculet;
TColor OutColor;

SizeKorrection(Input, Output);

CountX = Input->Picture->Bitmap->Width;
CountY = Input->Picture->Bitmap->Height;

for (i = 1; i < CountY - 1; i++)
for (j = 1; j < CountX - 1; j++)
{
A = PixelBright(Input->Canvas->Pixels[j - 1][i - 1]);
B = PixelBright(Input->Canvas->Pixels[j][i - 1]);
C = PixelBright(Input->Canvas->Pixels[j + 1][i - 1]);
D = PixelBright(Input->Canvas->Pixels[j - 1][i]);
E = PixelBright(Input->Canvas->Pixels[j + 1][i]);
F = PixelBright(Input->Canvas->Pixels[j - 1][i + 1]);
G = PixelBright(Input->Canvas->Pixels[j][i + 1]);
H = PixelBright(Input->Canvas->Pixels[j + 1][i + 1]);

Calc_G = sqrt(abs( ((F + 2*G + H)*(F + 2*G + H)) - ((A + 2*B + C)*(A
+ 2*B + C)) ));
Calc_V = sqrt(abs( ((A + 2*D + F)*(A + 2*D + F)) - ((C + 2*E + H)*(C
+ 2*E + H)) ));
Calculet = sqrt((Calc_G*Calc_G) + (Calc_V*Calc_V));

Calculet = Calculet / 3;
```

```
OutColor = (TColor) (Calculet + (Calculet * 256) + (Calculet * 256 * 256));
Output->Canvas->Pixels[j][i] = OutColor;
}

Artefact_3x3(Output);
}
```

Код, реализующий вызов функции Gradient_Sobel, показан на листинге 9.

Листинг 9 Пример программного кода, реализующего вызов функции Gradient_Sobel.

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Gradient_Sobel (Image1, Image2);
}
```

Код, реализующий обработку изображения оператором Лапласа, показан на листинге 10.

16

Листинг 10 Пример программного кода, реализующего обработку изображения оператором Лапласа.

```
void Gradient_Laplas(TImage *Input, TImage *Output, int PositivPorog, int NegativPogor)
{
int CountX, CountY;
int i, j;
int A,B,C,D,E;
float Calculet;
TColor OutColor;

SizeKorrection(Input, Output);

CountX = Input->Picture->Bitmap->Width;
CountY = Input->Picture->Bitmap->Height;

for (i = 1; i < CountY - 1; i++)
for (j = 1; j < CountX - 1; j++)
```

```
{
A = PixelBright(Input->Canvas->Pixels[j][i]);
B = PixelBright(Input->Canvas->Pixels[j][i - 1]);
C = PixelBright(Input->Canvas->Pixels[j - 1][i]);
D = PixelBright(Input->Canvas->Pixels[j + 1][i]);
E = PixelBright(Input->Canvas->Pixels[j][i + 1]);

Calculet = (B + C + D + E - 4*A);
if (Calculet > 0)
{  if (Calculet > PositivPorog)
   OutColor = clRed;
   else
   OutColor = clBlack; }
else
{  if (abs(Calculet) > NegativPogor)
   OutColor = clYellow;
   else
   OutColor = clBlack; }

Output->Canvas->Pixels[j][i] = OutColor;
}

Artefact_3x3(Output);
}
```

Код, реализующий вызов функции Gradient_Laplas, показан на листинге 11.

Листинг 11 Пример программного кода, реализующего вызов функции Gradient_Sobel.

```
void __fastcall TForm1::Button5Click(TObject *Sender)
{
Gradient_Laplas(Image1, Image2, 100, 100);
}
```

3. Задание на выполнение лабораторной работы и порядок её выполнения

Задание на выполнение лабораторной работы следует выбирать из таблицы 1 в соответствии с номером студента в списке группы.

Таблица 1 Задания на выполнение лабораторной работы

Вариант	Метод обработки	Комментарий
1	Оператор Робертса	Правая верхняя четверть
2	Градиент	Левая верхняя четверть
3	Оператор Робертса	Правая нижняя четверть
4	Градиент	Левая нижняя четверть
5	Оператор Собела	Верхняя половина
6	Оператор Лапласа	Нижняя половина
7	Оператор Собела	Правая верхняя четверть
8	Оператор Лапласа	Левая верхняя четверть
9	Оператор Робертса	Правая нижняя четверть
10	Градиент	Левая нижняя четверть

Первая часть лабораторной работы заключается в обработке изображения с использованием виртуального лабораторного комплекса Vision Lab. Для выполнения обработки следует выполнить следующую последовательность действий:

1. Запустить приложение «виртуальный лабораторный комплекс Vision Lab»
2. Загрузить изображение, которое будет подвергнуто обработке
3. Выбрать вкладку «Подчёркивание контуров»
4. Нажать кнопку «Градиент».
5. Сохранить результат обработки (результат должен быть представлен в отчёте)
6. Произвести обработку с использованием оператора Робертса. Для этого следует нажать кнопку «Оператор Робертса».
7. Сохранить результат обработки (результат должен быть представлен в отчёте)

8. Произвести обработку с использованием оператора Собела. Для этого следует нажать кнопку «Оператор Собела».
9. Сохранить результат обработки (результат должен быть представлен в отчёте)
10. Произвести обработку с использованием оператора Лапласа. Для этого следует нажать кнопку «Оператор Лапласа».
11. Сохранить результат обработки (результат должен быть представлен в отчёте)

В отчёте о выполнении лабораторной работы должен быть показан процесс работы с программой и результаты обработки изображения.

Вторая часть лабораторной работы заключается в реализации алгоритмов пороговой обработки на языке C++ в среде программирования C++ Builder.

Отчет о выполнении лабораторной работы должен содержать разработанный программный код и изображение программы во время работы. Также отчет должен содержать изображение до и после обработки.

При разработке программы, осуществляющую подчёркивание контуров, следует использовать метод обработки, рекомендованный в задании.

Рекомендуемая литература

1. Сойфер В.А. Методы компьютерной обработки изображений [Текст] / В.А. Сойфер, ФИЗМАТЛИТ. – Москва, 2003.
2. Фисенко В.Т., Фисенко Т.Ю. Компьютерная обработка и распознавание изображений: Учебное пособие. - СПб.: СПбГУ ИТМО, 2008. - 192 с.
3. Афонин В.Л., Макушкин В.А. Интеллектуальные робототехнические системы. - М.: Изд-во "Интернет-университет информационных технологий - ИНТУИТ.ру", 2005. - 208 с.: ил.

4. Архангельский А.Я. Программирование в C++Builder 4 (+ CD-ROM). - М.: Бином, 2000. - 1088 с.
5. Елманова Н.З., Кошель С.П. Введение в Borland C++Builder 4. - М.: Диалог-МИФИ, 2000. - 352 с.
6. Калверт Ч., Рейсдорф К. Borland C++ Builder 5. - Киев:"ДиаСофт", 2000. - 944 с.
7. Архангельский А.Я. Интегрированная среда разработки C++Builder 5. - М.: Бином, 2000. - 272 с.
8. Холингворт Дж., Сворт Б., Кэшмэн М., Густавсон П. Borland C++ Builder 6. Руководство разработчика. - М.: Издательский дом "Вильямс", 2003. - 976 с.
9. Архангельский А.Я. C++Builder 6. Справочное пособие. Книга 1. Язык C++. - М.: Бином, 2002. - 544 с.