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_____ 2021 г.

ОБРАБОТКА ИЗОБРАЖЕНИЙ В СИСТЕМАХ ТЕХНИЧЕСКОГО ЗРЕНИЯ

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

Курск 2021

УДК 621

Составители: А.В. Мальчиков

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

Обработка изображений в системах технического зрения: методические указания к выполнению практической и самостоятельной работы по дисциплине «Информационные системы роботов и обработка сигналов» / Юго-Зап. гос. ун-т; сост. А.В. Мальчиковв. Курск, 2021. 16 с.: ил. 7, табл. 1. Библиогр.: с. 15.

Методические указания содержат сведения о методах обработки изображений, в частности, методах наложения масок Кирша, 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 15.12.21 . Формат 60x84 1/16
Усл.печ.л. 0,9 . Уч.-изд.л. 0,8 Тираж 10 экз. Заказ 3151 Бесплатно.
Юго-Западный государственный университет.
305040 Курск, ул. 50 лет Октября, 94

ОБРАБОТКА ИЗОБРАЖЕНИЙ В СИСТЕМАХ ТЕХНИЧЕСКОГО ЗРЕНИЯ

Цель работы: *изучение приёмов разработки программ реализующих обработку изображений путём наложения масок Кирша, Лапласа, Певитта, Робертса, Робинсона.*

Объект исследования: *маски Кирша, Лапласа, Певитта, Робертса, Робинсона.*

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

1. Краткие теоретические сведения

Операция наложения маски широко применяется при обработке изображений. При использовании масок с небольшим числом элементов обработка требует относительно небольших вычислительных ресурсов. Вместе с тем с помощью различных масок можно производить широкий спектр преобразований изображений – размытие, подчёркивание контуров, создание эффекта объёма и другие.

В общем виде прямоугольную маску m×n можно представить следующим образом:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m,0} & a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} \quad (1)$$

Процедура умножения на маску выполняется следующим образом: каждый элемент маски умножается на определённый элемент изображения, причём элемент маски $a_{i,j}$ умножается на элемент изображения $X_{I+i,J+j}$, где I и J – индексы элемента изображения, который обрабатывается наложением маски. Это может быть записано следующим выражением:

$$Y_{I,J} = \left(a_{0,0} \cdot X_{I,J} + a_{0,1} \cdot X_{I,J+1} + ... + a_{0,n} \cdot X_{I,J+n}\right) + \left(a_{1,0} \cdot X_{I+1,J} + a_{1,1} \cdot X_{I+1,J+1} + ... + a_{0,n} \cdot X_{I+1,J+n}\right) + \\ + ... + \left(a_{m,0} \cdot X_{I+m,J} + a_{m,1} \cdot X_{I+m,J+1} + ... + a_{m,n} \cdot X_{I+m,J+n}\right)$$

На практике чаще всего применяются маски размерностью 3×3, 2×2 и меньшие. Достаточно часто маска 3×3 выглядит следующим образом:

$$\begin{pmatrix} a_{-1,-1} & a_{0,-1} & a_{1,-1} \\ a_{-1,0} & a_{0,0} & a_{1,0} \\ a_{-1,1} & a_{0,1} & a_{1,1} \end{pmatrix} \qquad (2)$$

Такое представление маски 3×3 было использовано в виртуальном лабораторном комплексе Vision Lab. Чтобы применить реализованный в виртуальном лабораторном комплексе Vision Lab метод наложения маски необходимо выбрать вкладку «Наложение маски» и нажать на кнопку «Умножение на маску 3 на 3», «Умножение на две маски 3 на 3» или «Умножение на маску 5 на 5». Лабораторный комплекс имеет встроенную библиотеку масок, в которую входят маски градиента, Кирша, Лапласа, Певитта, Робертса, Робинсона, сглаживания, рельефа, размытия водой и другие. Так же есть возможность использовать свои маски. Процедура умножения на 2 маски позволяет наложить на изображение 2 маски и сложить результаты. Это удобно для реализации градиентных методов обработки.

На рисунке 1 представлено изображение до и после наложения маски Кирша.

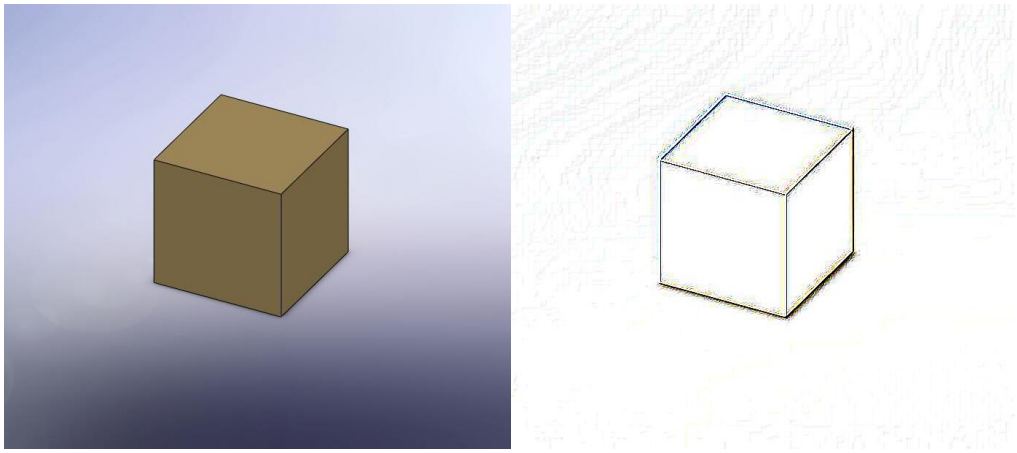

Рис. 1 Изображение до и после обработки маской Кирша; исходное изображение слева

2. Методика выполнения практической работы

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

Вторая часть практической 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 2.

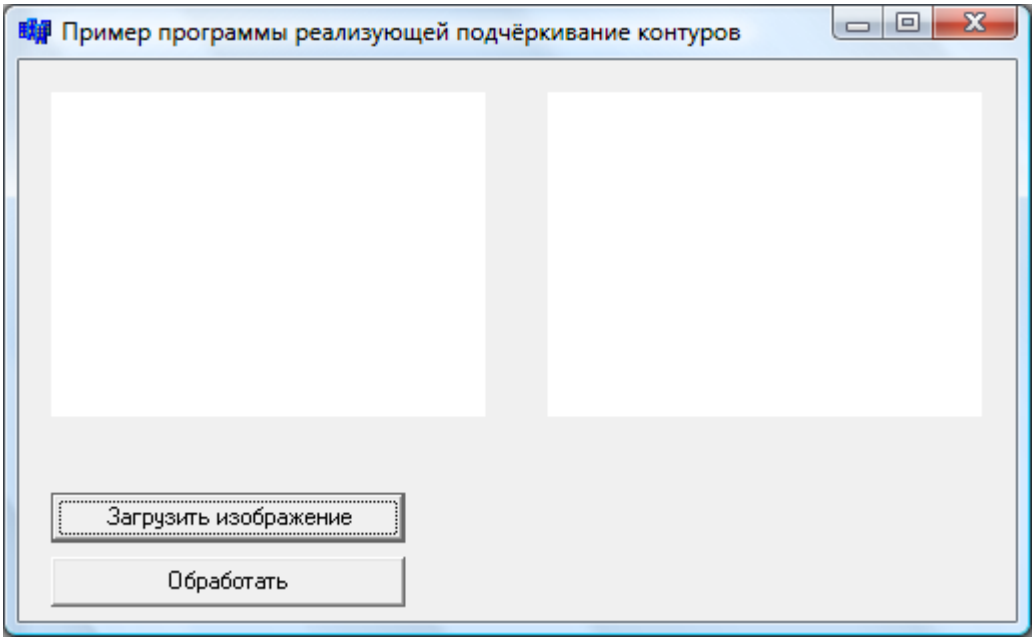

Рис. 2 Пример пользовательского интерфейса

Предложенный пример пользовательского интерфейса представляет собой четыре компонента из стандартной библиотеки C++ Builder – два компонента Image и два компонента Button. Компонент Image расположен на вкладке Additional панели компонентов (см. рисунок 3).

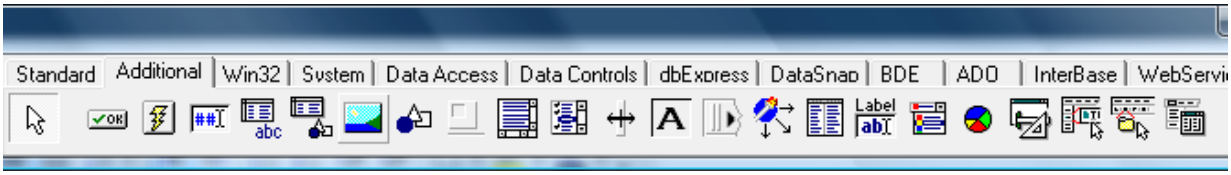

Рис. 3 Вкладка Additional панели компонентов C++ Builder

Компонент Button расположен на вкладке Standard панели компонентов (см. рисунок 4).

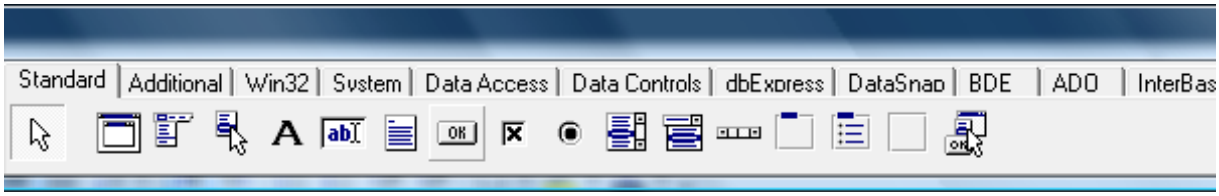

Рис. 4 Вкладка Standard панели компонентов C++ Builder

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 5.

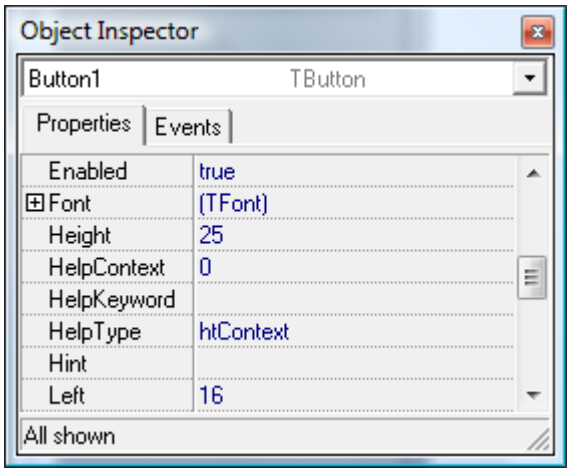

Рис. 5 Окно Object Inspector отображающее свойства компонента Button

В предложенном примере были изменены свойства Name для компонентов Button, свойства Proportional и Stretch компонентов Image и свойства, определяющие положение компонентов на форме. 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 6.

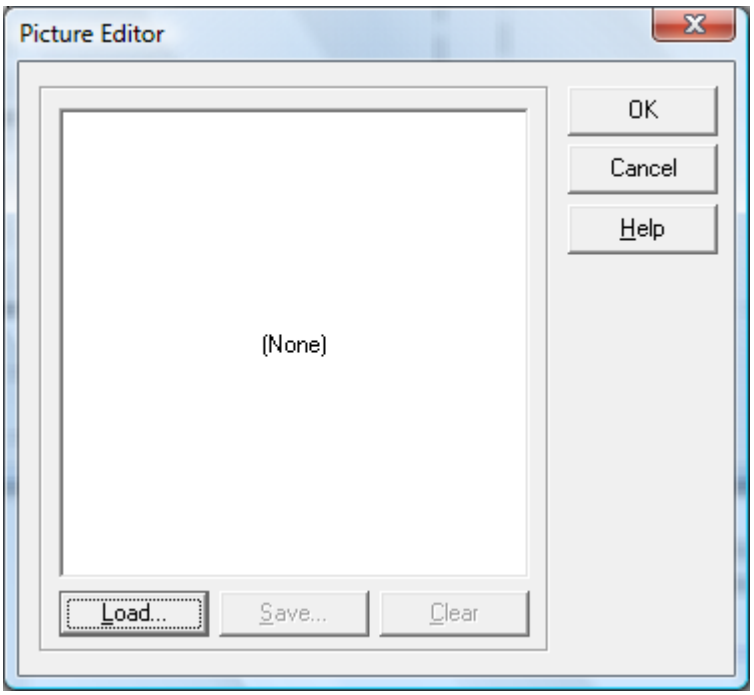

Рис. 6 Диалоговое окно Picture Editor

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

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 7).

Рис. 7 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);
}
```

Для организации алгоритма наложения маски на изображение 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;
}
```

Также для организации алгоритма наложения маски на изображение нужно разработать процедуру умножения участка изображения на маску. Данный код представлен на листинге 4.

Листинг 4 Пример программного кода, реализующего процедуру умножения участка изображения на маску.

```
int MatrixMultiplicat(Matrix3 A, Matrix3 B, float Delitel)
{
int C;

C = A.a11*B.a11 + A.a12*B.a12 + A.a13*B.a13 +
   A.a21*B.a21 + A.a22*B.a22 + A.a23*B.a23 +
   A.a31*B.a31 + A.a32*B.a32 + A.a33*B.a33;

C = C / Delitel;

if (C < 0) C = 0;
if (C > 255) C = 255;

return C;
}
```

В приведенных примерах используются некоторые нестандартные типы данных и глобальные переменные. Объявления данных типов данных и переменных показаны на листинге 5.

Листинг 5 Объявление нестандартных типов данных и глобальные переменные, использованные в примерах.

```
struct Matrix3 {
int a11, a12, a13, a21, a22, a23, a31, a32, a33;};
```

```
byte MasR [1200][1200];
byte MasG [1200][1200];
byte MasB [1200][1200];
```

Используя разработанные функции становится возможным организовать обработку изображения наложением маски. В примере показано использование маски Робертса.

Листинг 6 Пример функции, обрабатывающей изображение маской Робертса.

```
void Okno3(TImage *Input, TImage *Output)
{
int CountX, CountY;
int i, j;
TColor Color, OutColor;
int Blue, Green, Red;
bool B;
Matrix3 A, Mask;
float Delitel;

SizeKorrection(Input, Output);

Mask.a11 = 0;
Mask.a12 = 0;
Mask.a13 = 0;
Mask.a21 = 0;
Mask.a22 = -1;
Mask.a23 = 0;
Mask.a31 = 0;
Mask.a32 = 0;
Mask.a33 = 1;
Delitel = 1;



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

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

```
for (j = 0; j < CountX; j++)
{
Color = Input->Canvas->Pixels[j][i];

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

MasB[j][i] = Blue;
MasG[j][i] = Green;
MasR[j][i] = Red;

}

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

A.a11 = MasB[j - 1][i - 1];
A.a12 = MasB[j][i - 1];
A.a13 = MasB[j + 1][i - 1];

A.a21 = MasB[j - 1][i];
A.a22 = MasB[j][i];
A.a23 = MasB[j + 1][i];

A.a31 = MasB[j - 1][i + 1];
A.a32 = MasB[j][i + 1];
A.a33 = MasB[j + 1][i + 1];

Blue = MatrixMultiplicat(A, Mask, Delitel);

A.a11 = MasG[j - 1][i - 1];
A.a12 = MasG[j][i - 1];
A.a13 = MasG[j + 1][i - 1];

A.a21 = MasG[j - 1][i];
A.a22 = MasG[j][i];
A.a23 = MasG[j + 1][i];
```

```
A.a31 = MasG[j - 1][i + 1];
A.a32 = MasG[j][i + 1];
A.a33 = MasG[j + 1][i + 1];

Green = MatrixMultiplicat(A, Mask, Delitel);

A.a11 = MasR[j - 1][i - 1];
A.a12 = MasR[j][i - 1];
A.a13 = MasR[j + 1][i - 1];

A.a21 = MasR[j - 1][i];
A.a22 = MasR[j][i];
A.a23 = MasR[j + 1][i];

A.a31 = MasR[j - 1][i + 1];
A.a32 = MasR[j][i + 1];
A.a33 = MasR[j + 1][i + 1];

Red = MatrixMultiplicat(A, Mask, Delitel);

OutColor = (TColor) (Blue*(256*256) + (Green*256) + Red);

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

Artefact_3x3(Output);
}
```

В приведённом примере были использованы две нестандартные функции. Код этих функций приведён на листинге 7.

Листинг 7 Использованные в приведённых примерах функции.

```
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_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];
}
```

3. Задание на выполнение самостоятельной работы и порядок её выполнения

Задание на выполнение самостоятельной 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	Маска Кирша	Левая верхняя четверть
2	Маска Лапласа	Правая нижняя четверть
3	Маска Певитта	Левая верхняя четверть
4	Маска Робертса	Нижняя половина
5	Маска Робинсона	Верхняя половина
6	Маска Лапласа	Левая нижняя четверть
7	Маска Певитта	Правая верхняя четверть
8	Маска Кирша	Правая верхняя четверть
9	Маска Робертса	Правая нижняя четверть
10	Маска Робинсона	Левая нижняя четверть

Первая часть 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 «Наложение маски»
4. Нажать кнопку «Умножение на маску 3 на 3».
5. В появившемся окне «Маска» ввести маску вручную или нажать кнопку «Загрузить маску» и выбрать файл маски.

6. Произвести обработку маской Кирша.
7. Сохранить результат обработки (результат должен быть представлен в отчёте)
8. Произвести обработку маской Лапласа.
9. Сохранить результат обработки (результат должен быть представлен в отчёте)
10. Произвести обработку маской Певитта.
11. Сохранить результат обработки (результат должен быть представлен в отчёте)
12. Произвести обработку маской Робертса.
13. Сохранить результат обработки (результат должен быть представлен в отчёте)
14. Произвести обработку маской Робинсона.
15. Сохранить результат обработки (результат должен быть представлен в отчёте)

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

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

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

При разработке программы, осуществляющей обработку изображения маской, 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.