Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет телекоммуникаций

Кафедра сетей и устройств телекоммуникаций

**К. А. Волков, В. Ю. Цветков, В. К. Конопелько**

# *АЛГОРИТМЫ И ПРОГРАММНЫЕ БИБЛИОТЕКИ ОБРАБОТКИ МУЛЬТИМЕДИЙНОЙ ИНФОРМАЦИИ*

*Рекомендовано УМО по образованию*
*в области информатики и радиоэлектроники*
*в качестве учебно-методического пособия для специальности*
*1-98 01 02 «Защита информации в телекоммуникациях»*
*и направления специальности 1-45 01 01-05*
*«Инфокоммуникационные технологии*
*(системы распределения мультимедийной информации)»*



Минск БГУИР 2017

УДК 004.032.6(076)
ББК 32.973.202-018.2я73
В67

Р е ц е н з е н т ы:

кафедра телекоммуникационных систем учреждения образования «Белорусская государственная академия связи» (протокол №6 от 05.01.2015);

заместитель директора по учебной и информационно-аналитической работе филиала Белорусского национального технического университета «Институт повышения квалификации и переподготовки кадров по новым направлениям развития техники, технологии и экономики БНТУ», кандидат технических наук, доцент И. А. Тавгень

**Волков, К. А.**

В67 Алгоритмы и программные библиотеки обработки мультимедийной информации : учеб.-метод. пособие / К. А. Волков, В. Ю. Цветков, В. К. Конопелько. – Минск : БГУИР, 2017. – 99 с. : ил.

ISBN 978-985-543-246-4.

Рассмотрены базовые алгоритмы обработки мультимедийной информации и программная библиотека OpenCV для их реализации.

**УДК 004.032.6(076)**
**ББК 32.973.202-018.2я73**

**ISBN 978-985-543-246-4**

## Содержание

## Введение

OpenCV – это библиотека, разработанная в Центре разработки программного обеспечения Intel и включающая набор типов данных, функций и классов, реализующих алгоритмы обработки изображений и компьютерного зрения [1]. OpenCV написана на языке высокого уровня (C/C++) и включает алгоритмы интерпретации изображений, калибровки камеры по эталону, устранения оптических искажений, определения сходства, анализа перемещения объекта, определения формы объекта и слежения за объектом, 3D-реконструкции, сегментации объекта, распознавания жестов и т. д. [2–4]. Эта библиотека популярна за счёт открытости и возможности бесплатного использования в учебных коммерческих целях.

В разделе 1 представлены основы OpenCV. В разделе 2 рассмотрены базовые функции библиотеки. Раздел 3 посвящен основным функциям OpenCV. В нем содержатся примеры программ, являющиеся одновременно заданиями для выполнения на лабораторных и практических занятиях. Раздел 4 представляет указания по выполнению лабораторных работ.

# 1. Основы OpenCV

## 1.1. Структура OpenCV

Основные модули библиотеки OpenCV (рис. 1.1) [1]:

- **cxcore** – ядро: содержит базовые структуры данных и алгоритмы (базовые операции над многомерными числовыми массивами; матричная алгебра, математические функции, генераторы случайных чисел; запись/восстановление структур данных в/из XML; базовые функции 2D-графики);

- **CV –** модуль обработки изображений и компьютерного зрения (базовые операции над изображениями – фильтрация, геометрические преобразования, преобразование цветовых пространств и т. д.); анализ изображений (выбор отличительных признаков, морфология, поиск контуров, гистограммы); анализ движения, слежение за объектами; обнаружение объектов, в частности лиц; калибровка камер, элементы восстановления пространственной структуры);

- **Highgui** – модуль для ввода/вывода изображений и видео, создания пользовательского интерфейса (захват видео с камер и из видеофайлов, чтение/запись статических изображений; функции для организации простого UI (все демо-приложения используют HighGUI);

- **Cvaux –** экспериментальные и устаревшие функции (пространственное зрение: стереокалибровка, самокалибровка; поиск стереосоответствия, клики в графах; нахождение и описание черт лица);

- **CvCam –** захват видео (позволяет осуществлять захват видео с цифровых видеокамер (поддержка прекращена, в последних версиях этот модуль отсутствует).

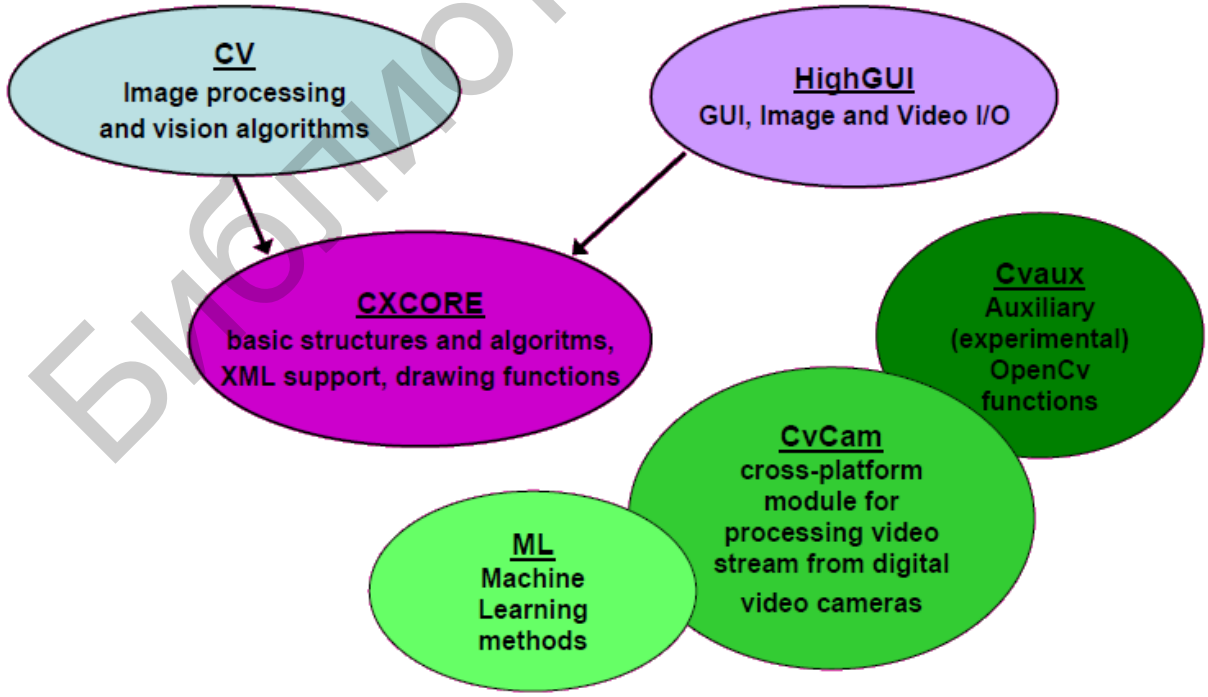


Рис. 1.1. Структура библиотеки OpenCV

В версии 2.2 структура библиотека реорганизована. Вместо больших универсальных модулей (cxcore, Cvaux, Highgui, Cvaux) библиотека OpenCV разделена на небольшие модули по функциональному использованию:

- **opencv_core –** ядро: содержит базовые структуры, вычисления (математические функции, генерация псевдослучайных чисел, DFT, DCT, ввод-вывод в XML и т. п.);
- **opencv_imgproc –** обработка изображений (фильтры, преобразования и т. д.);
- **opencv_highgui –** простой UI, загрузка/сохранение изображений и видео;
- **opencv_ml –** методы и модели машинного обучения (SVM, деревья принятия решений и т. д.);
- **opencv_features2d –** различные дескрипторы (SURF, SIFT и т. д.);
- **opencv_video** – анализ движения и отслеживание объектов (оптический поток, шаблоны движения, устранение фона);
- **opencv_objdetect** – детектирование объектов на изображении (вейвлеты Хаара, HOG и т. д.);
- **opencv_calib3d** – калибровка камеры, поиск стереосоответствия и элементы обработки 3-мерных данных;
- **opencv_flann** – библиотека быстрого поиска ближайших соседей (FLANN);
- **opencv_contrib** – код, не готовый для применения;
- **opencv_legacy** – устаревший код, сохраняемый для обратной совместимости;
- **opencv_gpu** – ускорение некоторых функций OpenCV за счет CUDA (NVidia).

## 1.2. Установка OpenCV

Запустив инсталлятор, необходимо выбрать директорию установки.

После установки библиотеки необходимо настроить рабочую среду для использования OpenCV (для VC++ 6 и версии OpenCV 1.0-1.1: Tools->Options, вкладка Directories; для VS C++ 2005/2008: Tools->Options->Projects and Solutions -> VC++ Directories). Для этого из списка выберите «Include Files» («Подключаемые файлы»), щелкните на иконке New и добавьте строчки (рис. 1.2):

"C:\Program Files\OpenCV\cv\include"
"C:\Program Files\OpenCV\cxcore\include"
"C:\Program Files\OpenCV\otherlibs\highgui"
"C:\Program Files\OpenCV\cvaux\include"
"C:\Program Files\OpenCV\otherlibs\cvcam\include"
"C:\Program Files\OpenCV\ml\include"
"C:\Program Files\OpenCV\otherlibs_graphics\include"
"C:\Program Files\OpenCV\otherlibs_graphics\include\jasper"

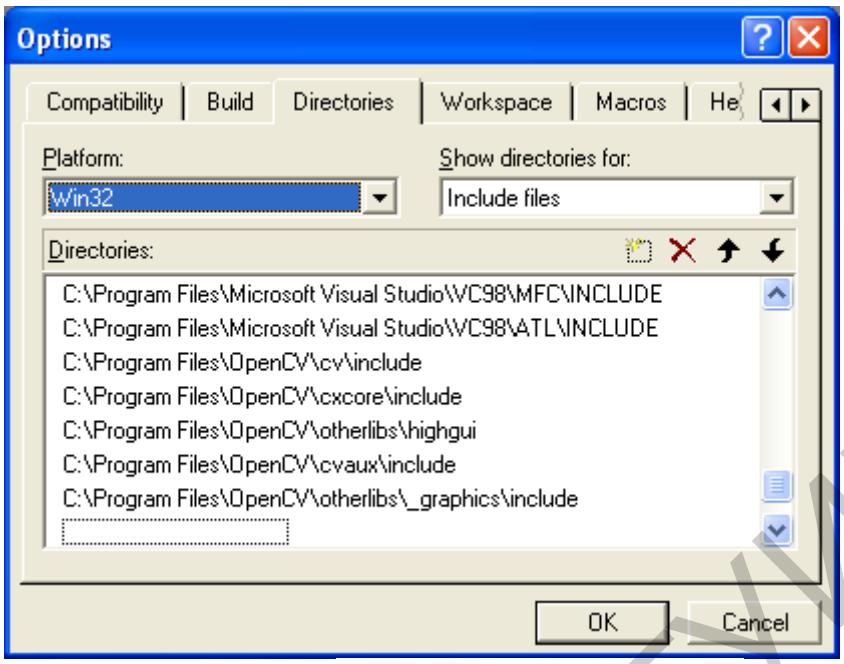


Рис. 1.2. Окно настройки рабочей среды

Далее выберите в списке «Library files» («Файлы библиотек») и добавьте строчки:

"C:\Program Files\OpenCV\lib"
"C:\Program Files\OpenCV\otherlibs_graphics\lib"

Далее в том же списке выберите «Source Files» («Файлы исходного кода») и добавьте строчки:

"C:\Program Files\OpenCV\cv\src"
"C:\Program Files\OpenCV\cxcore\src"
"C:\Program Files\OpenCV\cvaux\src"
"C:\Program Files\OpenCV\otherlibs\highgui"
"C:\Program Files\OpenCV\otherlibs\cvcam\src\windows"
"C:\Program Files\OpenCV\ml\src"
"C:\Program Files\OpenCV\otherlibs_graphics\src"

При создании нового проекта необходимо выполнить следующее:

- в открытом проекте выберите: Project->Properties (Проект->Свойства);
- выберите Configuration Properties->Linker->Input;
- добавьте имена библиотек в поле «Additional Dependencies» (дополнительные зависимости): **cxcore.lib cv.lib highgui.lib cvaux.lib cvcam.lib;**
- пропишите путь к каталогу с dll-файлами в PATH: C:\Program Files\OpenCV\bin;
- для VC++ 2008 и версии OpenCV 2.0:

include:
c:\DevTools\OpenCV2.0\include\opencv\

library:
c:\DevTools\OpenCV2.0\lib\Release

Source files:
c:\DevTools\OpenCV2.0\src\cv\
c:\DevTools\OpenCV2.0\src\cvaux\
c:\DevTools\OpenCV2.0\src\cxcore\
c:\DevTools\OpenCV2.0\src\highgui\
c:\DevTools\OpenCV2.0\src\ml\

Каталог Release по умолчанию не существует. Поэтому необходимо:

- установить Cmake для Windows;
- перейти в каталог OpenCV: c:\DevTools\OpenCV2.0;
- выполнить команду cmake . -G "Visual Studio 9 2008" (вместо Visual Studio 9 2008 может быть использован другой компилятор).

Cmake сгенерирует solution-файл (содержащий все проекты) для Visual Studio. Откройте его и выполните build с выбором Release-версии. Далее добавьте пути:

Tools–>Options–>Projects–>VC++ Directories–>Include files:
c:\DevTools\OpenCV2.0\include\opencv

Tools–>Options–>Projects–>VC++ Directories–>Library files:
c:\DevTools\OpenCV2.0\lib\Release

Tools–>Options–>Projects–>VC++ Directories–>Source files:
c:\DevTools\OpenCV2.0\src\cv\
c:\DevTools\OpenCV2.0\src\cvaux\
c:\DevTools\OpenCV2.0\src\cxcore\
c:\DevTools\OpenCV2.0\src\highgui\
c:\DevTools\OpenCV2.0\src\ml\

При создании нового проекта добавьте библиотеки для линковки «Project–>Properties–>Linker->Input–>Additional dependencies» **cxcore200.lib cv200.lib highgui200.lib cvaux200.lib**.

Установка версии OpenCV 2.1 аналогична установке версии 2.0 и даже проще, т. к. у неё уже в комплекте идут скомпилированные lib-файлы OpenCV2.1\lib\. Необходимо линковать библиотеки **cxcore210.lib cv210.lib highgui210.lib cvaux210.lib.**

## 2. Реализация базовых функций с помощью OpenCV

### 2.1. Ввод-вывод текста

Далее приведен код для вывода на экран текста в окне [1]:

```
#include <cv.h>
#include <highgui.h>

int main( int argc, char** argv )
{
	// задаём высоту и ширину картинки
	int height = 620;
	int width = 440;
	// задаём точку для вывода текста
	CvPoint pt = cvPoint( height/4, width/2 );
	// Создаём 8-битную 3-канальную картинку
	IplImage* hw = cvCreateImage(cvSize(height, width), 8, 3);
	// заливаем картинку чёрным цветом
	cvSet(hw,cvScalar(0,0,0));
	// инициализация шрифта
	CvFont font;
	cvInitFont( &font, CV_FONT_HERSHEY_COMPLEX,1.0, 1.0, 0, 1, CV_AA);
	// используя шрифт, выводим на картинку текст
	cvPutText(hw, "OpenCV Step By Step", pt, &font, CV_RGB(150, 0, 150) );

	// создаём окно
	cvNamedWindow("Hello World", 0);
	// показываем картинку в созданном окне
	cvShowImage("Hello World", hw);
	// ждём нажатия клавиши
	cvWaitKey(0);

	// освобождаем ресурсы
	cvReleaseImage(&hw);
	cvDestroyWindow("Hello World");
	return 0;
}
```

Приведённый пример выводит окно с надписью «OpenCV Step By Step» и после нажатия любой клавиши завершает работу.

Рассмотрим некоторые функции.

Функция

```
int cvNamedWindow( const char* name, int flags );
```

создаёт окно, в которое выводится изображение. Первый параметр – название окна, второй – его размер (0 – даёт возможность изменять размер окна во время выполнения программы, но обычно указывается флаг **CV_WINDOW_AUTOSIZE**, который означает, что окно будет ровно тех же размеров, что и загружаемое в него изображение).

Функция

void cvShowImage( const char* name, const CvArr* image );

отображает изображение в окне (первый параметр – название окна для вывода картинки, второй параметр – изображение для вывода).

Функция

int cvWaitKey( int delay=0 );

останавливает программу и ожидает нажатия клавиши заданное число миллисекунд и продолжает программу, если ничего не нажато. Если же, как в рассматриваемом случае, параметр функции равен нулю, тогда программа ожидает нажатия клавиши и только потом продолжает работу.

Стандартный пример использования этой функции в цикле обработки видеоданных с камеры:

```
char c = cvWaitKey(33);
if (c == 27)
{ // если нажали ESC – выходим из цикла
   break;
}
```

Функция

void cvReleaseImage( IplImage** image );

освобождает память, выделенную под изображение, и устанавливает указатель в NULL.

Функция

void cvDestroyWindow( const char* name );

закрывает окно и освобождает выделенную память.

## 2.2. Загрузка и вывод изображения

Рассмотрим простой пример, когда на экран выводится изображение из файла, имя которого передано в виде первого параметра [1].

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* src = 0;
```

```
int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);
	// клонируем картинку
	src = cvCloneImage(image);

	printf("[i] image: %s\n", filename);
	assert( src != 0 );

	// окно для отображения картинки
	cvNamedWindow("original",CV_WINDOW_AUTOSIZE);

	// показываем картинку
	cvShowImage("original",image);

	// выводим в консоль информацию о картинке
	printf( "[i] channels:  %d\n",        image->nChannels );
	printf( "[i] pixel depth: %d bits\n",   image->depth );
	printf( "[i] width:       %d pixels\n", image->width );
	printf( "[i] height:      %d pixels\n", image->height );
	printf( "[i] image size:  %d bytes\n",  image->imageSize );
	printf( "[i] width step:  %d bytes\n",  image->widthStep );

	// ждём нажатия клавиши
	cvWaitKey(0);

	// освобождаем ресурсы
	cvReleaseImage(& image);
	cvReleaseImage(&src);
	// удаляем окно
	cvDestroyWindow("original");
	return 0;
}
```

Рассмотрим некоторые функции из данного примера.

Функция

IplImage* cvLoadImage( const char* filename, int iscolor=CV_LOAD_IMAGE_COLOR – загружает картинку из файла, где filename – имя файла; iscolor – определяет как представить картинку; iscolor > 0 – цветная картинка с 3 каналами,

если iscolor == 0 – картинка будет загружена в формате GRAYSCALE (градации серого), если iscolor < 0 – картинка будет загружена как есть:

```
/* 8bit, color or not */
#define CV_LOAD_IMAGE_UNCHANGED  -1
/* 8bit, gray */
#define CV_LOAD_IMAGE_GRAYSCALE   0
/* ?, color */
#define CV_LOAD_IMAGE_COLOR       1
/* any depth, ? */
#define CV_LOAD_IMAGE_ANYDEPTH    2
/* ?, any color */
#define CV_LOAD_IMAGE_ANYCOLOR    4
```

Функция поддерживает следующие форматы изображений:
- Windows bitmaps - BMP, DIB;
- JPEG files - JPEG, JPG, JPE;
- Portable Network Graphics – PNG;
- Portable image format - PBM, PGM, PPM;
- Sun rasters - SR, RAS;
- TIFF files - TIFF, TIF.

Функция

```
IplImage* cvCloneImage( const IplImage* image );
```

делает полную копию изображения image, включая заголовок, данные и ROI (Region Of Interest – регион интересов – интересующая область на рисунке).

Обе функции возвращают указатель на изображение IplImage. Изображение в OpenCV представлено структурой вида

```
// OpenCV2.0\include\opencv\cxtypes.h
typedef struct _IplImage
{
   int  nSize;         /* sizeof(IplImage) */
   int  ID;            /* version (=0)*/
   int  nChannels;     /* Most of OpenCV functions support 1,2,3 or 4 channels */
   int  alphaChannel;  /* Ignored by OpenCV */
   int  depth;         /* Pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S,
IPL_DEPTH_16S,
                       IPL_DEPTH_32S, IPL_DEPTH_32F and IPL_DEPTH_64F are
supported.  */
   char colorModel[4];  /* Ignored by OpenCV */
   char channelSeq[4];  /* ditto */
   int  dataOrder;      /* 0 - interleaved color channels, 1 - separate color channels.
                       cvCreateImage can only create interleaved images */
   int  origin;        /* 0 - top-left origin,
```

```
                    1 - bottom-left origin (Windows bitmaps style).  */
    int  align;          /* Alignment of image rows (4 or 8).
                    OpenCV ignores it and uses widthStep instead.    */
    int  width;          /* Image width in pixels.                   */
    int  height;         /* Image height in pixels.                  */
    struct _IplROI *roi;    /* Image ROI. If NULL, the whole image is selected. */
    struct _IplImage *maskROI;      /* Must be NULL. */
    void  *imageId;              /* "          " */
    struct _IplTileInfo *tileInfo;  /* "          " */
    int  imageSize;         /* Image data size in bytes
                    (==image->height*image->widthStep
                    in case of interleaved data)*/
    char *imageData;        /* Pointer to aligned image data.         */
    int  widthStep;         /* Size of aligned image row in bytes.    */
    int  BorderMode[4];     /* Ignored by OpenCV.                     */
    int  BorderConst[4];    /* Ditto.                                 */
    char *imageDataOrigin;  /* Pointer to very origin of image data
                    (not necessarily aligned) -
                    needed for correct deallocation */
}
IplImage;
```

Таким образом, в консоли получаем следующую информацию:

image->nChannels // число каналов картинки (RGB, хотя в OpenCV - BGR ) (1–4)
image->depth // глубина в битах
image->width // ширина картинки в пикселях
image->height // высота картинки в пикселях
image->imageSize // память, занимаемая картинкой (==image->height*image->widthStep)
image->widthStep // расстояние между соседними по вертикали точками изображения (число байт в одной строчке картинки – может потребоваться для самостоятельного обхода всех пикселей изображения).

### 2.3. Загрузка и вывод видео

Вывод видео в OpenCV не сложнее, чем вывод одной картинки, только в данном случае потребуется использовать цикл для последовательного вывода кадров. Также необходимо предусмотреть условия выхода из цикла для прекращения вывода видео. Работа с видео начинается с функции cvCreateFileCapture, которая предоставляет доступ к видеофайлу. Далее в бесконечном цикле последовательно получаем кадры видео один за другим с по-

мощью функции cvQueryFrame, которая возвращает указатель на структуру картинки IplImage [1].

Кадр выводится в окне, а затем с помощью функции cvWaitKey организуется ожидание 33 мс и затем цикл продолжается, если не нажата клавиша ESC. Значение 33 выбрано исходя из задержки, которая даёт возможность просмотра 30 кадров в секунду.

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* frame =0;

int main(int argc, char* argv[])
{
	// имя файла задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "test.avi";

	printf("[i] file: %s\n", filename);

	// окно для отображения картинки
	cvNamedWindow("original",CV_WINDOW_AUTOSIZE);

	// получаем информацию о видеофайле
	CvCapture* capture = cvCreateFileCapture( filename );

	while(1){
		// получаем следующий кадр
		frame = cvQueryFrame( capture );
		if( !frame ) {
			break;
		}

		// здесь можно вставить
		// процедуру обработки

		// показываем кадр
		cvShowImage( "original", frame );

		char c = cvWaitKey(33);
		if (c == 27) { // если нажата ESC – выходим
			break;
		}
```

```
    }

    // освобождаем ресурсы
    cvReleaseCapture( &capture );
    // удаляем окно
    cvDestroyWindow("original");
    return 0;
}
```

Функция
CVAPI(CvCapture*) cvCreateFileCapture( const char* filename );
принимает название файла для считывания и возвращает указатель на структуру CvCapture, которая содержит информацию о видеофайле.

Функция
CVAPI(IplImage*) cvQueryFrame( CvCapture* capture );
получает и возвращает кадр с камеры или из файла. В качестве параметра получает указатель на структуру CvCapture. При этом возвращаемое изображение не должно освобождаться или модифицироваться.

Функция
CVAPI(void) cvReleaseCapture( CvCapture** capture );
освобождает память, связанную со структурой CvCapture.

### 2.4. Подключение линейки прокрутки

При просмотре видео полезна линейка прокрутки. Для её подключения используются функции пользовательского интерфейса HighGUI. Ползунок привязывается к конкретному окну функцией cvCreateTrackbar(). У неё в качестве последнего параметра передаётся указатель на функцию-обработчик, которая вызывается при изменении положения ползунка. Для установки положения на интересующем кадре используется функция cvSetCaptureProperty(), которая устанавливает параметры видеозахвата [1].

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

CvCapture* capture =0;
IplImage* frame =0;

//
// функция-обработчик ползунка
// перематывает на нужный кадр
void myTrackbarCallback(int pos) {
```

```
	cvSetCaptureProperty(capture, CV_CAP_PROP_POS_FRAMES, pos);
}

int main(int argc, char* argv[])
{
	// имя файла задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "test.avi";

	printf("[i] file: %s\n", filename);

	// окно для отображения картинки
	cvNamedWindow("original",CV_WINDOW_AUTOSIZE);

	// получаем информацию о видеофайле
	capture = cvCreateFileCapture( filename );

	// получаем число кадров
	double framesCount = cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_COUNT);
	printf("[i] count: %.0f\n", framesCount);
	int frames = (int)framesCount;

	int currentPosition=0;
	if( frames!= 0 ) {
		// показываем ползунок
		cvCreateTrackbar("Position", "original", ¤tPosition, frames,
myTrackbarCallback);
	}

	while(1){
		// получаем следующий кадр
		frame = cvQueryFrame( capture );
		if( !frame ) {
			break;
		}

		// здесь можно вставить
		// процедуру обработки

		// показываем кадр
		cvShowImage( "original", frame );

		char c = cvWaitKey(33);
		if (c == 27) { // если нажата ESC – выходим
```

```
                break;
            }
        }

        // освобождаем ресурсы
        cvReleaseCapture( &capture );
        // удаляем окно
        cvDestroyWindow("original");
        return 0;
}
```

Функции

CVAPI(double) cvGetCaptureProperty( CvCapture* capture, int property_id );
CVAPI(int) cvSetCaptureProperty( CvCapture* capture, int property_id, double value );

обеспечивают получение и установку свойства видеозахвата, где capture – указатель на видеоструктуру; property_id – идентификатор свойств:

```
#define CV_CAP_PROP_POS_MSEC       0 // текущее положение в миллисекундах
#define CV_CAP_PROP_POS_FRAMES     1 // начинающийся от 0 индекс кадра, который должен быть захвачен следующим
#define CV_CAP_PROP_POS_AVI_RATIO  2 // относительное положение видеофайла 0 – начало, 1 – конец
#define CV_CAP_PROP_FRAME_WIDTH    3 // ширина изображения
#define CV_CAP_PROP_FRAME_HEIGHT   4 // высота изображения
#define CV_CAP_PROP_FPS            5 // частота кадров
#define CV_CAP_PROP_FOURCC         6 // 4-буквенный код кодека
#define CV_CAP_PROP_FRAME_COUNT    7
#define CV_CAP_PROP_FORMAT         8
#define CV_CAP_PROP_MODE           9
#define CV_CAP_PROP_BRIGHTNESS    10 // яркость изображения (только для камеры)
#define CV_CAP_PROP_CONTRAST      11 // контрастность изображения (только для камеры)
#define CV_CAP_PROP_SATURATION    12 // насыщенность изображения (только для камеры)
#define CV_CAP_PROP_HUE           13 // тон изображения (только для камеры)
#define CV_CAP_PROP_GAIN          14
#define CV_CAP_PROP_EXPOSURE      15
#define CV_CAP_PROP_CONVERT_RGB   16
#define CV_CAP_PROP_WHITE_BALANCE 17
#define CV_CAP_PROP_RECTIFICATION 18
```

value – устанавливаемая величина.

Функция

CVAPI(int) cvCreateTrackbar( const char* trackbar_name, const char* window_name, int* value, int count, CvTrackbarCallback on_change);
создаёт ползунок и отображает его в заданном окне, где trackbar_name – название ползунка; window_name – название окна, в котором будет отображаться ползунок; value – указатель на целочисленную переменную, которая хранит текущую позицию ползунка; count – максимальная позиция ползунка (минимальная всегда 0); on_change – указатель на функцию, которая будет вызываться каждый раз при изменении положения ползунка.

Функция-обработчик должна иметь вид
void (CV_CDECL *CvTrackbarCallback2)(int pos, void* userdata);
где pos – текущее положение ползунка.

Дополнительные функции:
CVAPI(int) cvGetTrackbarPos( const char* trackbar_name, const char* window_name );
CVAPI(void) cvSetTrackbarPos( const char* trackbar_name, const char* window_name, int pos );
обеспечивают получение и установку положения ползунка,
где trackbar_name – название ползунка; window_name – название окна; pos – положение.

### 2.5. Видеозахват

Работа с камерой незначительно отличается от работы с видеофайлами. Вместо функции cvCreateFileCapture() в этом случае необходимо использовать функцию cvCreateCameraCapture(), которая в качестве параметра принимает не название файла, а идентификатор камеры [1].

Далее приводится программа, представляющая собой удобную утилиту для работы с камерой. Программа подключается и начинает захват с камеры с помощью cvCreateCameraCapture(), далее получает ширину и высоту кадра с помощью cvGetCaptureProperty(), затем в цикле при помощи cvQueryFrame() получает картинку с камеры и выводит в окне. При нажатии клавиши ESC программа выходит из цикла и завершается, а при нажатии клавиши Enter текущий кадр сохраняется в файл ImageN.jpg, где N – номер кадра, начиная от 0 (Image0.jpg, Image1.jpg и т. д.).

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
    // получаем любую подключённую камеру
    CvCapture* capture = cvCreateCameraCapture(CV_CAP_ANY);
//cvCaptureFromCAM( 0 );
```

```
    assert( capture );

    //cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH,
640);//1280);
    //cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT,
480);//960);

    // узнаем ширину и высоту кадра
    double width = cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH);
    double height = cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT);
    printf("[i] %.0f x %.0f\n", width, height );

    IplImage* frame=0;

    cvNamedWindow("capture", CV_WINDOW_AUTOSIZE);

    printf("[i] press Enter for capture image and Esc for quit!\n\n");

    int counter=0;
    char filename[512];

    while(true){
        // получаем кадр
        frame = cvQueryFrame( capture );

        // показываем
        cvShowImage("capture", frame);

        char c = cvWaitKey(33);
        if (c == 27) { // нажата ESC
            break;
        }
        else if(c == 13) { // Enter
            // сохраняем кадр в файл
            sprintf(filename, "Image%d.jpg", counter);
            printf("[i] capture... %s\n", filename);
            cvSaveImage(filename, frame);
            counter++;
        }
    }
    // освобождаем ресурсы
    cvReleaseCapture( &capture );
```

```
    cvDestroyWindow("capture");
    return 0;
}
```

OpenCV v1.0 показывает и сохраняет картинку с минимальным разрешением камеры 320×240 пикселей. Установка большего разрешения при помощи cvSetCaptureProperty() не поддерживается.

Функция

```
#define cvCaptureFromCAM cvCreateCameraCapture
CVAPI(CvCapture*) cvCreateCameraCapture( int index );
```

начинает захват с камеры,

где index – номер камеры в системе (состоит из суммы порядкового номера и так называемого домена);

0 – первая попавшаяся камера (можно использовать, если работает всего одна камера);

возможные домены:

```
#define CV_CAP_ANY      0     // autodetect
#define CV_CAP_MIL      100   // MIL proprietary drivers
#define CV_CAP_VFW      200   // platform native
#define CV_CAP_V4L      200
#define CV_CAP_V4L2     200
#define CV_CAP_FIREWARE 300   // IEEE 1394 drivers
#define CV_CAP_FIREWIRE 300
#define CV_CAP_IEEE1394 300
#define CV_CAP_DC1394   300
#define CV_CAP_CMU1394  300
#define CV_CAP_STEREO   400   // TYZX proprietary drivers
#define CV_CAP_TYZX     400
#define CV_TYZX_LEFT    400
#define CV_TYZX_RIGHT   401
#define CV_TYZX_COLOR   402
#define CV_TYZX_Z       403
#define CV_CAP_QT       500   // QuickTime
#define CV_CAP_UNICAP   600   // Unicap drivers
#define CV_CAP_DSHOW    700   // DirectShow (via videoInput)
```

Функция

```
CVAPI(int) cvSaveImage( const char* filename, const CvArr* image, const int* params
CV_DEFAULT(0);
```

сохраняет изображение в файл,

где filename – имя файла;

image – изображение для сохранения.

При успешном сохранении функция вернёт 1, а при неудаче – 0.

## 2.6. Запись видео в файл

OpenCV предоставляет возможность не только просматривать видео, но также и сохранять его в файл. Для этого необходимо инициализировать структуру редактора с помощью функции cvCreateVideoWriter(), в качестве параметров которой передаются название файла для сохранения видео, 4-буквенный код видеокодека, размер кадра и частота кадров. Далее при получении кадра с камеры (или очередного изображения из набора) он записывается в файл с помощью функции cvWriteFrame(). В конце необходимо освободить редактор функцией cvReleaseVideoWriter(). Приведённый далее фрагмент выполняет данные процедуры [1].

```
// Программа получает видео с камеры и записывает в avi файл

#include <cv.h>
#include <highgui.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
	cvNamedWindow("capture");

	// получаем любую подключённую камеру
	CvCapture *capture = cvCreateCameraCapture(CV_CAP_ANY);
	assert(capture!=0);

	const char *filename = "capture.avi";

	// частота кадров
	//double fps = cvGetCaptureProperty (capture, CV_CAP_PROP_FPS);
	double fps = 15;

	// размер картинки
	//CvSize size = cvSize( (int)cvGetCaptureProperty( capture,
CV_CAP_PROP_FRAME_WIDTH), (int)cvGetCaptureProperty( capture,
CV_CAP_PROP_FRAME_HEIGHT));
	CvSize size = cvSize(640, 480);

	CvVideoWriter *writer = cvCreateVideoWriter(filename,
CV_FOURCC('X','V','I','D'), fps, size, 0);
	assert(writer!=0);

	IplImage *frame=0;
```

```
        while(true){
            // получаем кадр
            frame = cvQueryFrame( capture );

            // сохраняем в файл
            cvWriteFrame(writer, frame);

            // показываем
            cvShowImage("capture", frame);
            char c = cvWaitKey(1);
            if (c == 27) { // если нажата ESC – выходим
                break;
            }
        }

        // освобождаем ресурсы
        cvReleaseCapture( &capture );
        cvReleaseVideoWriter(&writer);
        cvDestroyWindow("capture");
        return 0;
}
```

Заданный кодек установлен в системе.

Функция

CVAPI(CvVideoWriter*) cvCreateVideoWriter( const char* filename, int fourcc, double fps, CvSize frame_size, int is_color CV_DEFAULT(1)); инициализирует запись видео в файл и возвращает указатель на структуру редактора видеофайла:

```
struct CvVideoWriter
{
   virtual ~CvVideoWriter() {}
   virtual bool writeFrame(const IplImage*) { return false; }
};
```

где filename – имя файл для записи;

fourcc – 4-буквенный код кодека для обработки видео, формируется макросом CV_FOURCC:

```
CV_FOURCC('X','V','I','D') - кодек XviD
CV_FOURCC('P','I','M','1')    = MPEG-1
CV_FOURCC('M','J','P','G')    = motion-jpeg  (does not work well)
CV_FOURCC('M', 'P', '4', '2') = MPEG-4.2
CV_FOURCC('D', 'I', 'V', '3') = MPEG-4.3
CV_FOURCC('D', 'I', 'V', 'X') = MPEG-4
CV_FOURCC('U', '2', '6', '3') = H263
CV_FOURCC('I', '2', '6', '3') = H263I
```

CV_FOURCC ('F', 'L', 'V', '1') = FLV1
fps – частота кадров созданного видеопотока;
frame_size – размер кадра;
is_color – определяет, сохранять изображение в цвете (1) или в градациях серого(0) (пока только под Windows).

Функция
CVAPI(int) cvWriteFrame( CvVideoWriter* writer, const IplImage* image );
записывает/добавляет кадр в видеофайл,
где writer – указатель на структуру редактора видеофайла; image – изображение для сохранения.

Функция
CVAPI(void) cvReleaseVideoWriter( CvVideoWriter** writer );
закрывает видеофайл,
где writer – указатель на структуру редактора видеофайла.

## 2.7. Обработка событий

Модуль HighGUI предоставляет функции пользовательского интерфейса обрабатывать события, полученные как от указателя, так и от мыши. Обработчик мыши привязывается с помощью функции cvSetMouseCallback(), причём привязка осуществляется к конкретному окну. В приводимом ниже коде события от мыши обрабатываются функцией myMouseCallback(), которая при щелчке левой кнопкой мыши (CV_EVENT_LBUTTONDOWN) вызывает функцию drawTarget(), которая рисует красный круг и перекрестие из двух линий [1].

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* src = 0;

// рисуем целеуказатель
void drawTarget(IplImage* img, int x, int y, int radius)
{
    cvCircle(img,cvPoint(x, y),radius,CV_RGB(250,0,0),1,8);
    cvLine(img, cvPoint(x-radius/2, y-radius/2), cvPoint(x+radius/2,
y+radius/2),CV_RGB(250,0,0),1,8);
    cvLine(img, cvPoint(x-radius/2, y+radius/2), cvPoint(x+radius/2, y-
radius/2),CV_RGB(250,0,0),1,8);
}

// обработчик событий от мыши
```

```
void myMouseCallback( int event, int x, int y, int flags, void* param )
{
	IplImage* img = (IplImage*) param;
	switch( event ){
		case CV_EVENT_MOUSEMOVE:
			break;

		case CV_EVENT_LBUTTONDOWN:
			printf("%d x %d\n", x, y);
			drawTarget(img, x, y, 10);
			break;

		case CV_EVENT_LBUTTONUP:
			break;
	}
}

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);
	// клонируем картинку
	src = cvCloneImage(image);

	printf("[i] image: %s\n", filename);
	assert( src != 0 );

	// окно для отображения картинки
	cvNamedWindow("original",CV_WINDOW_AUTOSIZE);

	// вешаем обработчик мыши
	cvSetMouseCallback( "original", myMouseCallback, (void*) image);

	while(1){
		// показываем картинку
		cvCopyImage( image, src );
		cvShowImage( "original", src );

		char c = cvWaitKey(33);
		if (c == 27) { // если нажата ESC – выходим
			break;
		}
```

```
    }
    // освобождаем ресурсы
    cvReleaseImage(&image);
    cvReleaseImage(&src);
    // удаляем окно
    cvDestroyWindow("original");
    return 0;
}
```

Функция
void cvSetMouseCallback( const char* window_name, CvMouseCallback on_mouse, void* param=NULL );
устанавливает обработчик на события от мыши,
где window_name – название окна, к которому привязывается событие;
on_mouse – указатель на функцию-обработчик, которая будет вызываться для обработки события от мыши над заданным окном;
param – дополнительный параметр, который можно передать функции-обработчику (в примере – это указатель на изображение).

Функция-обработчик должна иметь вид
typedef void (CV_CDECL *CvMouseCallback )(int event, int x, int y, int flags, void* param);
где event – событие для обработки:

```
#define CV_EVENT_MOUSEMOVE      0
#define CV_EVENT_LBUTTONDOWN    1
#define CV_EVENT_RBUTTONDOWN    2
#define CV_EVENT_MBUTTONDOWN    3
#define CV_EVENT_LBUTTONUP      4
#define CV_EVENT_RBUTTONUP      5
#define CV_EVENT_MBUTTONUP      6
#define CV_EVENT_LBUTTONDBLCLK  7
#define CV_EVENT_RBUTTONDBLCLK  8
#define CV_EVENT_MBUTTONDBLCLK  9
```

x, y – координаты положения указателя в координатах изображения;
flags – флаг события:

```
#define CV_EVENT_FLAG_LBUTTON   1
#define CV_EVENT_FLAG_RBUTTON   2
#define CV_EVENT_FLAG_MBUTTON   4
#define CV_EVENT_FLAG_CTRLKEY   8
#define CV_EVENT_FLAG_SHIFTKEY  16
#define CV_EVENT_FLAG_ALTKEY    32
```

param – параметр, который может задействовать пользователь (в примере – указатель на картинку);
cvCopyImage – оболочка для cvCopy:

```
#define cvCopyImage( src, dst )        cvCopy( src, dst, 0 )
```

Функция
CVAPI(void) cvCopy( const CvArr* src, CvArr* dst, const CvArr* mask CV_DEFAULT(NULL) );
копирует массив src в массив dst,
где mask – маска (8-битный 1-канальный массив), с помощью которой можно задать область для копирования т. е. в dst будут скопированы только элементы src, для которых маска не равна нулю).

Функция
void cvLine(CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color, int thickness=1, int line type=8, int shift=0 );
рисует отрезок линии между двумя точками,
где img – картинка для рисования;
pt1 – начальная точка отрезка;
pt2 – конечная точка отрезка;
color – цвет линии, можно задать макросом CV_RGB(r, g, b);
thickness – толщина линии;
type – тип линии (4-связанная, 8-связанная или CV_AA – сглаживание);
shift – сдвиг.

Функция
void cvCircle(CvArr* img, CvPoint center, int radius, CvScalar color, int thickness=1, int line type=8, int shift=0);
рисует круг,
где img – картинка для рисования;
center – центр круга;
radius – радиус круга;
color – цвет линии (можно задать макросом **CV_RGB(r, g, b))**;
thickness – толщина линии;
type – тип линии (4-связанная,8- связанная или CV_AA-сглаживание);
shift – сдвиг.

Связность, определяет число соседних пикселей, т. е. 4-связный – это только 4 пикселя: север, восток, юг и запад:
[ ][o][ ]
[o][x][o]
[ ][o][ ],
а у 8-связного учитываются пиксели по диагонали: северо-восточный, юго-восточный, юго-западный и северо-западный:
[o][o][o]
[o][x][o]
[o][o][o].

## 3. Обработка изображений с помощью OpenCV

### 3.1. Сглаживание изображения

Ниже рассмотрен модифицированный пример загрузки изображения, в котором добавлено дополнительное окно для вывода результата операции сглаживания, осуществляемой функцией cvSmooth(). В данном примере выполняется сглаживание по гауссиану (**CV_GAUSSIAN**) в области 3×3 вокруг каждого пикселя изображения. Эту функцию можно использовать для устранения шумов на изображении [1–4].

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* dst = 0;

int main(int argc, char* argv[])
{
    // имя картинки задаётся первым параметром
    char* filename = argc == 2 ? argv[1] : "Image0.jpg";
    // получаем картинку
    image = cvLoadImage(filename,1);
    // клонируем картинку
    dst = cvCloneImage(image);

    printf("[i] image: %s\n", filename);
    assert( image != 0 );

    // окно для отображения картинки
    cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("smooth",CV_WINDOW_AUTOSIZE);

    // сглаживаем исходную картинку
    cvSmooth(image, dst, CV_GAUSSIAN, 3, 3);

    // показываем картинку
    cvShowImage("original",image);
    cvShowImage("smooth",dst);

    // ждём нажатия клавиши
    cvWaitKey(0);
```

```
    // освобождаем ресурсы
    cvReleaseImage(&image);
    cvReleaseImage(&dst);
    // удаляем окно
    cvDestroyWindow("original");
    cvDestroyWindow("smooth");
    return 0;
}
```

Функция

```
CVAPI(void) cvSmooth( const CvArr* src, CvArr* dst,
                      int smoothtype CV_DEFAULT(CV_GAUSSIAN),
                      int size1 CV_DEFAULT(3),
                      int size2 CV_DEFAULT(0),
                      double sigma1 CV_DEFAULT(0),
                      double sigma2 CV_DEFAULT(0));
```

сглаживает картинку (например, для устранения шума), где src – исходная картинка для обработки; dst – картинка для сохранения результата операции; smoothtype – тип сглаживания:

```
#define CV_BLUR_NO_SCALE 0
#define CV_BLUR  1
#define CV_GAUSSIAN  2
#define CV_MEDIAN 3
#define CV_BILATERAL 4
```

CV_BLUR_NO_SCALE – суммирование без масштабирования size1×size2; CV_BLUR – суммирование с масштабированием 1/size1×size2; CV_GAUSSIAN – свёртка изображения с гауссовым ядром size1×size2; CV_MEDIAN – поиск среднего значения в окрестности size1×size2; CV_BILATERAL – двусторонняя фильтрация окрестности size1×size2 с цветовой сигмой = sigma1 и пространственной сигмой = sigma2 (для квадратной окрестности, т. е. size1 = size2).

Функции передают параметры сглаживания:

size1 – размер области сглаживания (чем больше, тем сильнее сглаживание); size2 – (при гауссиане можно не задавать (0), тогда он будет равен size1); sigma1 – при гауссиане задаёт параметр отклонения g; если 0 – рассчитывается из размера ядра по формуле:

g = 0.3*(n/2 - 1) + 0.8 , где n – размер ядра size1,

если параметр sigma1 отличен от 0, а size1=size2=0, то размер ядра рассчитывается из sigma1.

### 3.2. Изменение размеров изображения

Изменение размеров изображения в OpenCV реализуется функцией cvResize(). Далее приведена программа, которая загружает изображение, затем в цикле создаёт 4 изображения с размерами в 3 раза меньше (можно задать любой другой размер), чем у исходного, и сразу же выполняет cvResize() с разным типом интерполяции [1–4]:

```
// изменение размеров изображения
// при помощи cvResize()
//
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

// исходная
IplImage* image = 0;
IplImage* dst[4];

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);

	int i=0;

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// создание уменьшенных картинок (разный тип интерполяции)
	for(i=0;i<4; i++){
		dst[i] = cvCreateImage( cvSize(image->width/3, image->height/3), image-
>depth, image->nChannels );
		cvResize(image, dst[i], i);
	}

	// окно для отображения картинки
	cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
	cvShowImage("original",image);

	// показываем результат
```

```
    char buf[128];
    for(i=0;i<4; i++){
        cvNamedWindow( itoa(i, buf, 10) ,CV_WINDOW_AUTOSIZE);
        cvShowImage(itoa(i, buf, 10), dst[i]);
    }

    // ждём нажатия клавиши
    cvWaitKey(0);

    // освобождаем ресурсы
    cvReleaseImage(&image);
    for(i=0;i<4; i++){
        cvReleaseImage(&dst[i]);
    }

    // удаляем окна
    cvDestroyAllWindows();
    return 0;
}
```

Функция
CVAPI(void) cvResize( const CvArr* src, CvArr* dst, int interpolation
CV_DEFAULT( CV_INTER_LINEAR ));
обеспечивает изменение размеров (масштабирование) изображения,
где src – исходное изображение;
dst – изображение для сохранения результата;
interpolation – метод интерполяции:
#define CV_INTER_NN 0 // nearest-neigbor – интерполяция методом ближайшего соседа
#define CV_INTER_LINEAR 1 // билинейная интерполяция (по умолчанию)
#define CV_INTER_CUBIC 2 //
#define CV_INTER_AREA 3 // бикубическая интерполяция

### 3.3. Выделение области интереса

**ROI** (**Region Of Interest** – регион интересов) – интересующая область изображения, один из фундаментов OpenCV. ROI позволяет пользователю задать определённую прямоугольную область изображения (т. е. ROI всегда должна находиться внутри исходного изображения). Почти все функции должны поддерживать работу с ROI, т. е. работу с выделенной областью изображения, что полезно для ускорения работы алгоритмов. Таким образом, если интересует только определённая область изображения, можно её выделить и работать только с ней, не затрагивая всё изображение целиком [1–4].

OpenCV предоставляет следующие функции для работы с ROI.

Функция
CVAPI(void)  cvSetImageROI( IplImage* image, CvRect rect );
обеспечивает установку интересующей области рисунка (COI не меняется),
где image – указатель на изображение;
rect – прямоугольник интересующей области.

Функция
CVAPI(void)  cvResetImageROI( IplImage* image );
сбрасывает область интересов (и COI),
где image – указатель на изображение.

Функция
CVAPI(CvRect) cvGetImageROI( const IplImage* image );
возвращает область интересов изображения,
где image – указатель на изображение.

Если ROI не установлена, функция вернёт cvRect(0,0,image->width,image->height).

ROI можно использовать для вырезания части изображения или, наоборот, добавления изображения.

Далее приведён пример работы с ROI. Программа загружает изображение и по заданным координатам устанавливает ROI с помощью функции **cvSetImageROI**(). Затем вызывается функция cvAddS(), которая добавляет к элементам массива (пикселям изображения) заданную скалярную величину. Результатом является цветовое выделение квадрата ROI.

```
// пример работы с ROI
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;

int main(int argc, char* argv[])
{
    // имя картинки задаётся первым параметром
    char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
    // получаем картинку
    image = cvLoadImage(filename,1);

    printf("[i] image: %s\n", filename);
    assert( image != 0 );

    cvNamedWindow("origianl", CV_WINDOW_AUTOSIZE);
```

```
    cvNamedWindow("ROI", CV_WINDOW_AUTOSIZE);

    // задаём ROI
    int x = argc >= 3 ? atoi(argv[2]) : 40;
    int y = argc >= 4 ? atoi(argv[3]) : 20;
    int width = argc >= 5 ? atoi(argv[4]) : 100;
    int height = argc >= 6 ? atoi(argv[5]) : 100;
    // добавочная величина
    int add =  argc >= 7 ? atoi(argv[6]) : 200;

    cvShowImage( "origianl", image);
    // устанавливаем ROI
    cvSetImageROI(image, cvRect(x,y,width,height));
    cvAddS(image, cvScalar(add), image);
    // сбрасываем ROI
    cvResetImageROI(image);
    // показываем изображение
    cvShowImage("ROI", image);

    // ждём нажатия клавиши
    cvWaitKey(0);

    // освобождаем ресурсы
    cvReleaseImage( &image );
    cvDestroyAllWindows();
    return 0;
}
```

Функция
CVAPI(void)  cvAddS( const CvArr* src, CvScalar value, CvArr* dst,
const CvArr* mask CV_DEFAULT(NULL));
обеспечивает расчёт суммы массива и скаляра (скаляр добавляется к каждому элементу массива) по формуле
dst(I)=src(I)+value if mask(I)!=0,
где src – исходный массив;
value – скаляр для суммирования;
dst – массив для сохранения результата;
mask – маска (8-битный 1-канальный массив, определяющий, какие элементы массива подлежат изменению).

Далее более наглядный пример работы с ROI. Программа принимает четвертым параметром название изображения, которое накладывается в ROI на область первого изображения. Для этого область интересов сначала обнуляется с помощью cvZero(), а затем туда копируется содержимое массива второго изображения при помощи cvCopyImage().

```
// пример работы с ROI 2
// вывод в ROI другого изображения
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* src = 0;

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	cvNamedWindow("origianl", CV_WINDOW_AUTOSIZE);
	cvNamedWindow("ROI", CV_WINDOW_AUTOSIZE);

	// задаём ROI
	int x = argc >= 3 ? atoi(argv[2]) : 120;
	int y = argc >= 4 ? atoi(argv[3]) : 50;
	// добавочное изображение
	char* filename2 = argc >= 5 ? argv[4] : "eye.jpg";
	src = cvLoadImage(filename2,1);
	assert( src != 0 );

	// размер ROI
	int width = src->width;
	int height = src->height;
	cvShowImage( "origianl", image);
	// устанавливаем ROI
	cvSetImageROI(image, cvRect(x,y,width,height));
	// обнулим изображение
	cvZero(image);

	// копируем изображение
```

```
    cvCopyImage(src, image);
    // сбрасываем ROI
    cvResetImageROI(image);
    // показываем изображение
    cvShowImage("ROI", image);
    // ждём нажатия клавиши
    cvWaitKey(0);
    // освобождаем ресурсы
    cvReleaseImage( &image );
    cvReleaseImage( &src );
    cvDestroyAllWindows();
    return 0;
}
```

Функция

```
CVAPI(void)  cvSetZero( CvArr* arr );
#define cvZero  cvSetZero
```

очищает все элементы массива arr (устанавливает их в 0).

### 3.4. Морфологические преобразования

Основные морфологические преобразования: **Erode** – размытие (операция сужения) и **Dilate** – растяжение (операция расширения) OpenCV реализует двумя функциями [1–4].

Функция

```
CVAPI(void)  cvErode( const CvArr* src, CvArr* dst,
                IplConvKernel* element CV_DEFAULT(NULL),
                int iterations CV_DEFAULT(1) );
```

размывает изображение с использованием фильтра (ядра) один или несколько раз, если element == NULL, используется ядро 3×3 (изображение формируется из локальных минимумов, т. е. будут увеличиваться тёмные области).

Функция

```
CVAPI(void)  cvDilate( const CvArr* src, CvArr* dst,
                IplConvKernel* element CV_DEFAULT(NULL),
                int iterations CV_DEFAULT(1) );
```

растягивает изображение с использованием фильтра (ядра) один или несколько раз, если element == NULL, используется ядро 3×3 (изображение формируется из локальных максимумов, т. е. будут увеличиваться светлые области).

В функциях использованы следующие переменные:
// #define CV_DEFAULT(val) = val; src – исходное изображение; dst – получаемое изображение; element – структурирующий элемент (ядро) по умолчанию, NULL соответствует ядру 3×3.

Структура ядра:

```
typedef struct _IplConvKernel
```

```
{
   int  nCols;
   int  nRows;
   int  anchorX;
   int  anchorY;
   int *values;
   int  nShiftR;
}
IplConvKernel;
```

где iterations – число итераций (сколько раз повторить морфологическое преобразование).

Обе функции допускают, чтобы в качесте dst выступало исходное изображение src.

В cvErode() ядро накладывается на изображение и на месте центра ядра остаётся минимальное значение, лежащее под ядром (в случае cvDilate() – наоборот – максимальное).

Эрозия (размытие/сужение) изображения обычно используется для избавления от случайных вкраплений на изображении. Идея состоит в том, что вкрапления при размытии устранятся, тогда как крупные и соответственно более визуально-значимые регионы остаются.

Растяжение (расширение) также должно устранять шум и способствовать объединению областей изображения, которые были разделены шумом, тенями. Применение небольшого растяжения должно сплавить эти области в одну.

Морфологические операции чаще всего применяются над двоичными изображениями, которые получаются после порогового преобразования (thresholding).

Создание ядра произвольной формы осуществляется функцией cvCreateStructuringElementEx():

```
CVAPI(IplConvKernel*)  cvCreateStructuringElementEx( int cols, int  rows, int anchor_x, int  anchor_y, int shape, int* values CV_DEFAULT(NULL) );
```

где cols – число колонок ядра;
rows – число сток ядра;
anchor_x – относительное горизонтальное смещение центра ядра;
anchor_y – относительное вертикальное смещение якоря ядра;
shape – форма ядра:

```
#define  CV_SHAPE_RECT      0
#define  CV_SHAPE_CROSS     1
#define  CV_SHAPE_ELLIPSE   2
#define  CV_SHAPE_CUSTOM    100
```

В последнем случае форму определяет пользователь через переменную values, которая содержит маску, определяющую, какие соседние пиксели должны учитываться, где values – указатель на массив, в котором ненулевые элементы определяют значимые пиксели. Если values==NULL, все элементы считают-

ся ненулевыми. Этот параметр учитывается только в случае shape==CV_SHAPE_CUSTOM.

Функция
CVAPI(void) cvReleaseStructuringElement( IplConvKernel** element );
обеспечивает освобождение памяти, выделенной под структурирующий элемент (ядро).

Выбирая различную структуру ядра, можно решать различные задачи обработки изображений:

- подавление шумов;
- выделение границ объекта;
- выделение скелета объекта.

Рассмотрим пример.

```
// пример базовых морфологических преобразований
// cvErode() и cvDilate()
//
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* dst = 0;
IplImage* erode = 0;
IplImage* dilate = 0;

int radius = 1;
int radius_max=10;

//
// функция-обработчик ползунка
// радиус ядра
void myTrackbarRadius(int pos) {
	radius = pos;
}

int iterations = 1;
int iterations_max = 10;

//
// функция-обработчик ползунка
// число итераций
void myTrackbarIterations(int pos) {
	iterations = pos;
```

```
}

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);
	// клонируем картинку
	dst = cvCloneImage(image);
	erode = cvCloneImage(image);
	dilate = cvCloneImage(image);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// окно для отображения картинки
	cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("erode",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("dilate",CV_WINDOW_AUTOSIZE);

	cvCreateTrackbar("Radius", "original", &radius, radius_max,
myTrackbarRadius);
	cvCreateTrackbar("Iterations", "original", &iterations, iterations_max,
myTrackbarIterations);


	while(1){

		// показываем картинку
		cvShowImage("original",image);

		// создаём ядро
		IplConvKernel* Kern = cvCreateStructuringElementEx(radius*2+1,
radius*2+1, radius, radius, CV_SHAPE_ELLIPSE);

		// выполняем преобразования
		cvErode(image, erode, Kern, iterations);
		cvDilate(image, dilate, Kern, iterations);

		// показываем результат
		cvShowImage("erode",erode);
		cvShowImage("dilate",dilate);
```

```
            cvReleaseStructuringElement(&Kern);

            char c = cvWaitKey(33);
            if (c == 27) { // если нажата ESC – выходим
                  break;
            }
      }

      // освобождаем ресурсы
      cvReleaseImage(&image);
      cvReleaseImage(&dst);
      cvReleaseImage(&erode);
      cvReleaseImage(&dilate);
      // удаляем окно
      cvDestroyWindow("original");
      cvDestroyWindow("erode");
      cvDestroyWindow("dilate");
      return 0;
}
```

Ещё одна функция – **cvMorphologyEx**() – обеспечивает более сложные морфологические преобразования изображения

```
CVAPI(void)  cvMorphologyEx( const CvArr* src, CvArr* dst,
                  CvArr* temp, IplConvKernel* element,
                  int operation, int iterations CV_DEFAULT(1) );
```

где src – исходное изображение;
dst – изображение для сохранения результата;
temp – для промежуточного хранения результатов (размер изображения должен совпадать с размером исходного изображения) – требуется при определённом значении operation;
operation – определяет тип морфологического преобразования:

```
#define      CV_MOP_OPEN                2
#define      CV_MOP_CLOSE               3
#define      CV_MOP_GRADIENT            4
#define      CV_MOP_TOPHAT              5
#define      CV_MOP_BLACKHAT            6
```

CV_MOP_OPEN и CV_MOP_CLOSE – комбинации сужения и расширения: **CV_MOP_OPEN** – сначала сужается, а затем расширяется (обычно используется для подсчёта регионов на двоичном изображении); **CV_MOP_CLOSE** – сначала расширяется, а затем сужается (обычно используется для уменьшения шумовых выбросов на границах регионов).
**CV_MOP_GRADIENT**: gradient(src) = Dilate(src)–Erode(src). Результатом этой операции над двоичным изображением станет выделение периметров суще-

ствующих пятен. На картинке с градациями серого градиент покажет, как быстро меняется яркость.

CV_MOP_TOPHAT и CV_MOP_BLACKHAT: TopHat(src) = src–Open(src); BlackHat(src) = Close(src)–src – комбинации изоляции: **CV_MOP_TOPHAT** – изолирует яркие локальные пики; **CV_MOP_BLACKHAT** – изоляция тёмных регионов.

Рассмотрим примеры.

**Пример 1**

```
// пример работы cvMorphologyEx()
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* dst = 0;

int radius = 1;
int radius_max=10;

//
// функция-обработчик ползунка
// радиус ядра
void myTrackbarRadius(int pos) {
	radius = pos;
}

int iterations = 1;
int iterations_max = 10;

//
// функция-обработчик ползунка
// число итераций
void myTrackbarIterations(int pos) {
	radius = pos;
}

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "Image0.jpg";
```

```
    // получаем картинку
    image = cvLoadImage(filename,1);
    // клонируем картинку
    dst = cvCloneImage(image);

    printf("[i] image: %s\n", filename);
    assert( image != 0 );

    // окно для отображения картинки
    cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("morphology",CV_WINDOW_AUTOSIZE);

    cvCreateTrackbar("Radius", "original", &radius, radius_max,
myTrackbarRadius);
    cvCreateTrackbar("Iterations", "original", &iterations, iterations_max,
myTrackbarIterations);


    while(1){

        // показываем картинку
        cvShowImage("original",image);

        // создаём ядро
        IplConvKernel* Kern = cvCreateStructuringElementEx(radius*2+1,
radius*2+1, radius, radius, CV_SHAPE_ELLIPSE);

        // картинка для промежуточного хранения результатов cvCreateImage
        IplImage* Temp = 0;
        Temp = cvCreateImage(cvSize(image->width, image->height) ,
IPL_DEPTH_8U, 1);
        // выолняем преобразование
        cvMorphologyEx(image, dst, Temp, Kern, CV_MOP_OPEN, iterations);


        // показываем результат
        cvShowImage("morphology",dst);

        cvReleaseStructuringElement(&Kern);
        cvReleaseImage(&Temp);

        char c = cvWaitKey(33);
        if (c == 27) { // если нажата ESC – выходим
            break;
```

```
            }
        }

        // освобождаем ресурсы
        cvReleaseImage(&image);
        cvReleaseImage(&dst);
        // удаляем окна
        cvDestroyAllWindows();
        return 0;
}
```

**Пример 2**

```
// пример работы cvMorphologyEx()
//
// покажем все методы сразу
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* open = 0;
IplImage* close = 0;
IplImage* gradient = 0;
IplImage* tophat = 0;
IplImage* blackhat = 0;

int radius = 1;
int radius_max=10;

//
// функция-обработчик ползунка
// радиус ядра
void myTrackbarRadius(int pos) {
        radius = pos;
}

int iterations = 1;
int iterations_max = 10;

//
```

```
// функция-обработчик ползунка
// число итераций
void myTrackbarIterations(int pos) {
	radius = pos;
}

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);
	// клонируем картинку
	open = cvCloneImage(image);
	close = cvCloneImage(image);
	gradient = cvCloneImage(image);
	tophat = cvCloneImage(image);
	blackhat = cvCloneImage(image);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// окно для отображения картинки
	cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("CV_MOP_OPEN",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("CV_MOP_CLOSE",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("CV_MOP_GRADIENT",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("CV_MOP_TOPHAT",CV_WINDOW_AUTOSIZE);
	cvNamedWindow("CV_MOP_BLACKHAT",CV_WINDOW_AUTOSIZE);

	cvCreateTrackbar("Radius", "original", &radius, radius_max, myTrackbarRadi-
us);
	cvCreateTrackbar("Iterations", "original", &iterations, iterations_max,
myTrackbarIterations);

	while(1){

		// показываем картинку
		cvShowImage("original",image);

		// создаём ядро
		IplConvKernel* Kern = cvCreateStructuringElementEx(radius*2+1, radi-
us*2+1, radius, radius, CV_SHAPE_ELLIPSE);
```

```
            // картинка для промежуточного хранения результатов cvCreateImage
            IplImage* Temp = 0;
            Temp = cvCreateImage(cvSize(image->width, image->height) ,
IPL_DEPTH_8U, 1);
            // выолняем преобразования
            cvMorphologyEx(image, open, Temp, Kern, CV_MOP_OPEN, iterations);
            cvMorphologyEx(image, close, Temp, Kern, CV_MOP_CLOSE, itera-
tions);
            cvMorphologyEx(image, gradient, Temp, Kern, CV_MOP_GRADIENT,
iterations);
            cvMorphologyEx(image, tophat, Temp, Kern, CV_MOP_TOPHAT, itera-
tions);
            cvMorphologyEx(image, blackhat, Temp, Kern, CV_MOP_BLACKHAT,
iterations);


            // показываем результат
            cvShowImage("CV_MOP_OPEN",open);
            cvShowImage("CV_MOP_CLOSE",close);
            cvShowImage("CV_MOP_GRADIENT",gradient);
            cvShowImage("CV_MOP_TOPHAT",tophat);
            cvShowImage("CV_MOP_BLACKHAT",blackhat);

            cvReleaseStructuringElement(&Kern);
            cvReleaseImage(&Temp);

            char c = cvWaitKey(33);
            if (c == 27) { // если нажата ESC – выходим
                  break;
            }
      }

      // освобождаем ресурсы
      cvReleaseImage(&image);
      cvReleaseImage(&open);
      cvReleaseImage(&close);
      cvReleaseImage(&gradient);
      cvReleaseImage(&tophat);
      cvReleaseImage(&blackhat);
      // удаляем окна
      cvDestroyAllWindows();
      return 0;
}
```

Функция
CVAPI(void) cvDestroyAllWindows(void);
удаляет все окна.

### 3.5. Заливка части изображения

Функция **cvFloodFill()** используется, если требуется пометить или изолировать область изображения для дальнейшего анализа, т. е. эта функция реализует заливку области [1–4].

Функция

```
CVAPI(void)  cvFloodFill( CvArr* image, CvPoint seed_point,
                CvScalar new_val, CvScalar lo_diff
CV_DEFAULT(cvScalarAll(0)),
                CvScalar up_diff CV_DEFAULT(cvScalarAll(0)),
                CvConnectedComp* comp CV_DEFAULT(NULL),
                int flags CV_DEFAULT(4),
                CvArr* mask CV_DEFAULT(NULL));
```

заполняет связанную область заданным цветом. Начиная с заданной начальной точки и далее, определяя принадлежность области по близости значений соседних пикселей. Описание аргументов:

- image – входное изображение (1- или 3-канальное), 8-битное или 32F (изображение модифицируется функцией, если не установлен флаг CV_FLOODFILL_MASK_ONLY);
- seed_point – начальная точка;
- new_val – новое значение пикселей;
- lo_diff – максимальное нижнее значение разницы в яркости/цвете между наблюдаемым пикселем и одним из его соседей, чтобы добавить его в область (в случае 8-битного цветного изображения – это упакованное значение);
- up_diff – максимальное верхнее значение разницы в яркости/цвете между наблюдаемым пикселем и одним из его соседей, чтобы добавить его в область (в случае 8-битного цветного изображения – это упакованное значение);
- comp – указатель на структуру CvConnectedComp, которую заполняет функция, возвращая в неё информацию о заполненной области:

```
typedef struct CvConnectedComp
{
   double area;    /* площадь  */
   CvScalar value; /* усреднённый цвет области */
   CvRect rect;    /* ROI – область интереса */
   CvSeq* contour; /* граница */
}
CvConnectedComp;
```

где flags – флаг операции.

Младшие биты содержат значение связности – 4(по умолчанию) или 8, используемое функцией. Связность определяет, сколько соседних пикселей обрабатываются. Старшие биты могут быть 0 или комбинацией флагов:
#define CV_FLOODFILL_FIXED_RANGE (1 << 16)
#define CV_FLOODFILL_MASK_ONLY  (1 << 17)
**CV_FLOODFILL_FIXED_RANGE** – если установлен, используется разница между текущим и начальным пикселем, в противном случае – разница между соседними пикселями (плавающий диапазон);
**CV_FLOODFILL_MASK_ONLY** – если установлен, то функция не вносит изменения в изображение, но заполняет маску, которая в данном случае должна быть отлична от NULL);
mask – маска операции: 1-канальное 8-битное изображение, на 2 пикселя шире и длинее, чем изображение. Если отлично от NULL, то функция использует и обновляет маску. Заполнение не будет идти поверх ненулевых пикселей маски. Например, детектор границ может использоваться в качестве маски для остановки заливки на границах. Возможно использование одной маски в нескольких вызовах функции для уверенности, что заполненные области не перекроются.

Рассмотрим пример.

```
// пример использования функции cvFloodFill()
// для заливки области
// начальный пиксель области выбирается по щелчку мышью
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

// заливка области картинки цветом
void fill(IplImage* src, CvPoint seed, CvScalar color=CV_RGB(255, 0, 0))
{
    assert(src);
    CvConnectedComp comp;

    cvFloodFill( src, seed, color,
          cvScalarAll(10), // минимальная разность
          cvScalarAll(10), // максимальная разность
          &comp,
          CV_FLOODFILL_FIXED_RANGE + 8,
          0);

    // покажем площадь заливки
```

```
        printf("[filled area] %.2f\n", comp.area);
}

// обработчик событий от мыши
void myMouseCallback( int event, int x, int y, int flags, void* param )
{
        IplImage* img = (IplImage*) param;

        switch( event ){
                case CV_EVENT_MOUSEMOVE:
                        break;

                case CV_EVENT_LBUTTONDOWN:
                        printf("%d x %d\n", x, y);

                        // вызываем нашу функцию-обёртку вокруг cvFloodFill()
                        fill(img, cvPoint(x, y));

                        break;

                case CV_EVENT_LBUTTONUP:
                        break;
        }
}

int main(int argc, char* argv[])
{
        IplImage *src=0, *dst=0;

        // имя картинки задаётся первым параметром
        char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
        // получаем картинку
        src = cvLoadImage(filename, 1);

        printf("[i] image: %s\n", filename);
        assert( src != 0 );

        // покажем изображение
        cvNamedWindow( "original", 1 );
        // вешаем обработчик мыши
        cvSetMouseCallback( "original", myMouseCallback, (void*) src);

        while(1){
                // показываем картинку
```

```
            cvShowImage( "original", src );
            char c = cvWaitKey(33);
            if (c == 27) { // если нажата ESC – выходим
                  break;
            }
      }
      // освобождаем ресурсы
      cvReleaseImage(&src);
      cvReleaseImage(&dst);
      // удаляем окна
      cvDestroyAllWindows();
      return 0;
}
```

Функция работает при щелчке в верхнем левом углу и установленном флаге **CV_FLOODFILL_FIXED_RANGE**. Если этот флаг убрать, то при той же начальной точке будет закрашена почти вся картинка.

### 3.6. Альфа-смешивание

Альфа-канал (alpha channel) – это значение точно такое же, как цветовые компоненты (красный, зеленый и синий). Оно определяет степень прозрачности для каждого пикселя изображения. Однако OpenCV не умеет работать с 4-канальными изображениями RGBA (A – это и есть обозначение альфа-канала). При загрузке RGBA-изображения оно просто преобразуется в RGB. Однако OpenCV реализует операцию альфа-смешивания (alpha blending), т. е. наложения изображений друг на друга с целью создания эффекта частичной прозрачности. Этого эффекта можно добиться при использовании функции cvAddWeighted()[1–4].

Функция

```
CVAPI(void)  cvAddWeighted( const CvArr* src1, double alpha,
                            const CvArr* src2, double beta,
                            double gamma, CvArr* dst );
```

вычисляет взвешенную поэлементную сумму двух массивов по формуле dst = src1 * alpha + src2 * beta + gamma,
где src1 – первый исходный массив;
alpha – вес элементов первого массива;
src2 – второй исходный массив;
beta – вес элементов второго массива;
gamma – добавочная величина к каждой сумме;
dst – массив для сохранения результата.

Массивы должны иметь одинаковый тип и размер (или размер ROI).

Рассмотрим следующий пример. Загружаем два изображения, причём первое должно быть больше второго, т. к. на изображении image с помо-

щью cvSetImageROI() выделяется область интереса размером со второе изображение. Затем пиксели ROI первой картинки и пиксели второй картинки поэлементно складываются с весовыми коэффициентами alpha = 0.5 и beta = 0.5, и получается эффект альфа-смешивания.

```
// alpha blending
// взвешенная поэлементная сумма двух массивов – cvAddWeighted()
//
//  http://dasl.mem.drexel.edu/~noahKuntz/openCVTut2.html
//

#include <cv.h>
#include <highgui.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* templ = 0;
IplImage* dst = 0;

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// шаблон
	char* filename2 = argc >= 3 ? argv[2] : "templ.bmp";
	printf("[i] template: %s\n", filename2);

	templ = cvLoadImage(filename2,1);
	assert( templ != 0 );

	cvNamedWindow("origianl", CV_WINDOW_AUTOSIZE);
	cvNamedWindow("template", CV_WINDOW_AUTOSIZE);
	cvNamedWindow("res", CV_WINDOW_AUTOSIZE);

	dst = cvCloneImage(templ);

	// размер шаблона
	int width = templ->width;
```

```
    int height = templ->height;

    // оригинал и шаблон
    cvShowImage( "origianl", image);
    cvShowImage( "template", templ);

    int x = 0;
    int y = 0;
    // задаём весовые коэффициенты
    double alpha = 0.5;
    double beta = 0.5;
    // устанавливаем область интереса
    cvSetImageROI(image, cvRect(x,y,width,height));
    // взвешенная сумма
    cvAddWeighted(image, alpha, templ, beta, 0.0, dst);
    // освобождаем область интереса
    cvResetImageROI(image);

    // показываем результат
    cvShowImage( "res", dst);

    // ждём нажатия клавиши
    cvWaitKey(0);

    // освобождаем ресурсы
    cvReleaseImage( &image );
    cvReleaseImage( &templ );
    cvReleaseImage( &dst );
    cvDestroyAllWindows();
    return 0;
}
```

### 3.7. Пороговое преобразование

Для выбора пикселей выше (ниже, между) определённого порогового значения используется функция **cvThreshold()**[1–4]**.**

Функция

```
CVAPI(double)  cvThreshold( const CvArr*  src, CvArr*  dst,
                            double  threshold, double  max_value,
                            int threshold_type );
```

выполняет фиксированное пороговое преобразование для элементов массива,
где src – исходный массив (изображение) (1-канальное 8- или 32-битное);
dst – целевой массив, должен иметь тот же тип, что и src или 8-битный;
threshold – пороговая величина;

max_value – максимальное значение (используется совместно с CV_THRESH_BINARY и CV_THRESH_BINARY_INV);
threshold_type – тип порогового преобразования:
#define CV_THRESH_BINARY 0 /* value = value > threshold ? max_value : 0 */
#define CV_THRESH_BINARY_INV 1 /* value = value > threshold ? 0 : max_value */
#define CV_THRESH_TRUNC 2 /* value = value > threshold ? threshold : value */
#define CV_THRESH_TOZERO 3 /* value = value > threshold ? value : 0 */
#define CV_THRESH_TOZERO_INV 4 /* value = value > threshold ? 0 : value */
#define CV_THRESH_MASK 7
#define CV_THRESH_OTSU 8 /* use Otsu algorithm to choose the optimal threshold value;
combine the flag with one of the above CV_THRESH_* values */

Обычное пороговое преобразование не учитывает, что части объектов могут иметь различную яркость из-за разницы в освещённости. Это можно исправить, если использовать адаптивное пороговое преобразование, которое рассматривает значение не в одном пикселе, а в окрестности пикселя. Это значение может быть просто средним значением пикселей окрестности (т. е. все пиксели равнозначны) (CV_ADAPTIVE_THRESH_MEAN_C) либо пиксели окрестности умножаются на весовой коэффициент (взвешиваются) в соответствии с функцией, например с гауссовой (CV_ADAPTIVE_THRESH_GAUSSIAN_C).

Функция

CVAPI(void) cvAdaptiveThreshold( const CvArr* src, CvArr* dst, double max_value,
int adaptive_method CV_DEFAULT(CV_ADAPTIVE_THRESH_MEAN_C),
int threshold_type CV_DEFAULT(CV_THRESH_BINARY),
int block_size CV_DEFAULT(3),
double param1 CV_DEFAULT(5));

выполняет адаптивное пороговое преобразование для элементов массива,
где rc – исходное изображение;
dst – целевое изображение;
max_value – максимальное значение (используется совместно с CV_THRESH_BINARY и CV_THRESH_BINARY_INV);
adaptive_method – используемый адаптационный алгоритм:
#define CV_ADAPTIVE_THRESH_MEAN_C 0
#define CV_ADAPTIVE_THRESH_GAUSSIAN_C 1
threshold_type – тип порогового преобразования:
#define CV_THRESH_BINARY 0 /* value = value > threshold ? max_value : 0 */
#define CV_THRESH_BINARY_INV 1 /* value = value > threshold ? 0 : max_value */

block_size – размер окрестности (в пикселях), которая используется для расчёта порогового значения: 3, 5, 7 и т. д.;
param1 – параметр, зависящий от используемого метода.

Для методов CV_ADAPTIVE_THRESH_MEAN_C и CV_ADAPTIVE_THRESH_GAUSSIAN_C – это константа, вычитаемая из среднего или взвешенного значения (может быть отрицательной).

Рассмотрим пример.

```
// пример порогового преобразования
// cvThreshold() и cvAdaptiveThreshold()
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
	IplImage *src=0, *dst=0, *dst2=0;

	// имя картинки задаётся первым параметром
	char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	src = cvLoadImage(filename, 0);

	printf("[i] image: %s\n", filename);
	assert( src != 0 );

	// покажем изображение
	cvNamedWindow( "original", 1 );
	cvShowImage( "original", src );

	dst = cvCreateImage( cvSize(src->width, src->height), IPL_DEPTH_8U, 1);
	dst2 = cvCreateImage( cvSize(src->width, src->height), IPL_DEPTH_8U, 1);

	cvThreshold(src, dst, 50, 250, CV_THRESH_BINARY);
	cvAdaptiveThreshold(src, dst2, 250,
CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY, 7, 1);

	// показываем результаты
	cvNamedWindow( "cvThreshold", 1 );
	cvShowImage( "cvThreshold", dst);
	cvNamedWindow( "cvAdaptiveThreshold", 1 );
	cvShowImage( "cvAdaptiveThreshold", dst2);
```

```
    // ждём нажатия клавиши
    cvWaitKey(0);

    // освобождаем ресурсы
    cvReleaseImage(&src);
    cvReleaseImage(&dst);
    cvReleaseImage(&dst2);
    // удаляем окна
    cvDestroyAllWindows();
    return 0;

}
```

Аналогичный результат можно получить при использовании функции для выборки тех пикселей изображения, которые лежат в заданном интервале значений (могут как храниться в массиве cvInRange(), так и задаваться скалярами cvInRangeS()).

Функция

```
CVAPI(void) cvInRange( const CvArr* src, const CvArr* lower,
                       const CvArr* upper, CvArr* dst );
```

поэлементная проверка элементов массива (проверяет, что значения элементов массива лежат между значениями элементов двух других массивов) по формуле
dst(idx) = lower(idx) <= src(idx) < upper(idx),
где src – исходный массив;
lower – массив с нижней границей (включая);
upper – массив с верхней границей (не включая);
dst – массив для хранения результата (тип 8S или 8U).

Функция

```
CVAPI(void) cvInRangeS( const CvArr* src, CvScalar lower,
                        CvScalar upper, CvArr* dst );
```

проверяет, что элемент массива лежит между двух скаляров, по формуле
dst(idx) = lower <= src(idx) < upper.

Пример использования.

```
// пример выборки значений в заданном интервале
// cvInRangeS()
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>
```

```
int main(int argc, char* argv[])
{

      IplImage *src=0, *dst=0;

    // имя картинки задаётся первым параметром
    char* filename = argc >= 2 ? argv[1] : "Image0.jpg";

    // получаем картинку
    src = cvLoadImage(filename, 0);

    printf("[i] image: %s\n", filename);
    assert( src != 0 );

    // покажем изображение
    cvNamedWindow( "original", 1 );
    cvShowImage( "original", src );

    dst = cvCreateImage( cvSize(src->width, src->height), IPL_DEPTH_8U, 1);

    cvInRangeS(src, cvScalar(50), cvScalar(255), dst);

    // показываем результаты
    cvNamedWindow( "cvInRangeS", 1 );
    cvShowImage( "cvInRangeS", dst);

    // ждём нажатия клавиши
    cvWaitKey(0);

    // освобождаем ресурсы
    cvReleaseImage(&src);
    cvReleaseImage(&dst);

    // удаляем окна
    cvDestroyAllWindows();
    return 0;

}
```

Простой пример практического применения этих функций – выборка определённых областей изображения, которые отличаются от других яркостью (в случае изображения 1-канального изображения (в градациях серого)) или цветом (в случае цветного изображения).

### 3.8. Поиск объекта по цвету

Самый распространённый способ выделить объект – это использовать цвет. Цветовые пространства бывают линейные и нелинейные. К линейным относится **RGB.** Нелинейные: **HSV, LAB.** Удобство HSV состоит в том, что координаты выбраны с учетом человеческого восприятия: Hue (Тон), Saturation (Насыщенность), Value (Intensity) (Интенсивность) [1–4].

Рассмотрим программу, которая считывает картинку (в качестве первого параметра), разбивает её на слои (**cvSplit()**), над каждым из которых можно проделать пороговое преобразование (**cvInRangeS()**), причём значения интервалов удобным образом изменяются с помощью ползунков. Результирующее значение выводится в окошке «rgb and» и представляет собой логическое И – **cvAnd()** между получившимися пороговыми картинками.

```
// позволяет подобрать параметры
// Rmin, Rmax, Gmin, Gmax, Bmin, Bmax
// для выделения нужного объекта по цвету
//
// robocraft.ru
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* dst = 0;

// для хранения каналов RGB
IplImage* rgb = 0;
IplImage* r_plane = 0;
IplImage* g_plane = 0;
IplImage* b_plane = 0;
// для хранения каналов RGB после преобразования
IplImage* r_range = 0;
IplImage* g_range = 0;
IplImage* b_range = 0;
// для хранения суммарной картинки
IplImage* rgb_and = 0;

int Rmin = 0;
int Rmax = 256;
```

```
int Gmin = 0;
int Gmax = 256;

int Bmin = 0;
int Bmax = 256;

int RGBmax = 256;

//
// функции-обработчики ползунка
//
void myTrackbarRmin(int pos) {
	Rmin = pos;
	cvInRangeS(r_plane, cvScalar(Rmin), cvScalar(Rmax), r_range);
}

void myTrackbarRmax(int pos) {
	Rmax = pos;
	cvInRangeS(r_plane, cvScalar(Rmin), cvScalar(Rmax), r_range);
}

void myTrackbarGmin(int pos) {
	Gmin = pos;
	cvInRangeS(g_plane, cvScalar(Gmin), cvScalar(Gmax), g_range);
}

void myTrackbarGmax(int pos) {
	Gmax = pos;
	cvInRangeS(g_plane, cvScalar(Gmin), cvScalar(Gmax), g_range);
}

void myTrackbarBmin(int pos) {
	Bmin = pos;
	cvInRangeS(b_plane, cvScalar(Bmin), cvScalar(Bmax), b_range);
}

void myTrackbarBmax(int pos) {
	Bmax = pos;
	cvInRangeS(b_plane, cvScalar(Bmin), cvScalar(Bmax), b_range);
}

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
```

```
char* filename = argc == 2 ? argv[1] : "Image0.jpg";
// получаем картинку
image = cvLoadImage(filename,1);

printf("[i] image: %s\n", filename);
assert( image != 0 );

// создаём картинки
rgb = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 3 );
r_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
g_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
b_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
r_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
g_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
b_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
rgb_and = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
//  копируем
cvCopyImage(image, rgb);
// разбиваем на отельные каналы
cvSplit( rgb, b_plane, g_plane, r_plane, 0 );

//
// определяем минимальное и максимальное значение
// у каналов HSV
double framemin=0;
double framemax=0;

cvMinMaxLoc(r_plane, &framemin, &framemax);
printf("[R] %f x %f\n", framemin, framemax );
Rmin = framemin;
Rmax = framemax;

cvMinMaxLoc(g_plane, &framemin, &framemax);
printf("[G] %f x %f\n", framemin, framemax );
Gmin = framemin;
Gmax = framemax;

cvMinMaxLoc(b_plane, &framemin, &framemax);
printf("[B] %f x %f\n", framemin, framemax );
Bmin = framemin;
Bmax = framemax;

// окна для отображения картинки
cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
```

```
cvNamedWindow("R",CV_WINDOW_AUTOSIZE);
cvNamedWindow("G",CV_WINDOW_AUTOSIZE);
cvNamedWindow("B",CV_WINDOW_AUTOSIZE);
cvNamedWindow("R range",CV_WINDOW_AUTOSIZE);
cvNamedWindow("G range",CV_WINDOW_AUTOSIZE);
cvNamedWindow("B range",CV_WINDOW_AUTOSIZE);
cvNamedWindow("rgb and",CV_WINDOW_AUTOSIZE);

cvCreateTrackbar("Rmin", "R range", &Rmin, RGBmax, myTrackbarRmin);
cvCreateTrackbar("Rmax", "R range", &Rmax, RGBmax, myTrackbarRmax);
cvCreateTrackbar("Gmin", "G range", &Gmin, RGBmax, myTrackbarGmin);
cvCreateTrackbar("Gmax", "G range", &Gmax, RGBmax, myTrackbarGmax);
cvCreateTrackbar("Bmin", "B range", &Gmin, RGBmax, myTrackbarBmin);
cvCreateTrackbar("Bmax", "B range", &Gmax, RGBmax, myTrackbarBmax);

//
// разместим окна по рабочему столу
//
if(image->width <1920/4 && image->height<1080/2){
	cvMoveWindow("original", 0, 0);
	cvMoveWindow("R", image->width+10, 0);
	cvMoveWindow("G", (image->width+10)*2, 0);
	cvMoveWindow("B", (image->width+10)*3, 0);
	cvMoveWindow("rgb and", 0, image->height+30);
	cvMoveWindow("R range", image->width+10, image->height+30);
	cvMoveWindow("G range", (image->width+10)*2, image->height+30);
	cvMoveWindow("B range", (image->width+10)*3, image->height+30);
}

while(true){

	// показываем картинку
	cvShowImage("original",image);

	// показываем слои
	cvShowImage( "R", r_plane );
	cvShowImage( "G", g_plane );
	cvShowImage( "B", b_plane );

	// показываем результат порогового преобразования
	cvShowImage( "R range", r_range );
	cvShowImage( "G range", g_range );
	cvShowImage( "B range", b_range );
```

```
            // складываем
            cvAnd(r_range, g_range, rgb_and);
            cvAnd(rgb_and, b_range, rgb_and);

            // показываем результат
            cvShowImage( "rgb and", rgb_and );

            char c = cvWaitKey(33);
            if (c == 27) { // если нажата ESC – выходим
                  break;
            }
      }
      printf("\n[i] Results:\n" );
      printf("[i][R] %d : %d\n", Rmin, Rmax );
      printf("[i][G] %d : %d\n", Gmin, Gmax );
      printf("[i][B] %d : %d\n", Bmin, Bmax );

      // освобождаем ресурсы
      cvReleaseImage(& image);
      cvReleaseImage(&rgb);
      cvReleaseImage(&r_plane);
      cvReleaseImage(&g_plane);
      cvReleaseImage(&b_plane);
      cvReleaseImage(&r_range);
      cvReleaseImage(&g_range);
      cvReleaseImage(&b_range);
      cvReleaseImage(&rgb_and);

      // удаляем окна
      cvDestroyAllWindows();
      return 0;
}
```

Функция

```
#define cvCvtPixToPlane cvSplit
CVAPI(void)  cvSplit( const CvArr* src, CvArr* dst0, CvArr* dst1,
                  CvArr* dst2, CvArr* dst3 );
```

разделяет многоканальный массив на 1-канальные или выделяет отдельные цветовые плоскости,
где src – исходный массив (изображение);
dst0...dst3 – массивы (изображения) для сохранения результатов.

Функция

```
#define cvCvtPlaneToPix cvMerge
CVAPI(void)  cvMerge( const CvArr* src0, const CvArr* src1,
```

const CvArr* src2, const CvArr* src3,
CvArr* dst );

выполняет обратную операцию – объединяет несколько 1-канальных массивов в один многоканальный массив,
где src0...src2 – исходные массивы (изображения);
dst – массив (изображение) для сохранения результатов.

Функция

CVAPI(void) cvAnd( const CvArr* src1, const CvArr* src2,
CvArr* dst, const CvArr* mask CV_DEFAULT(NULL));

выполняет поэлементую операцию конъюнкции (логического И (AND)) над элементами двух массивов,
где src1 – первый исходный массив (изображение) (если NULL, то считается, что массив состоит из 1);
src2 – второй исходный массив(изображение);
dst – массив (изображение) для сохранения результата;
mask – маска (8-битный 1-канальный массив).

Результат операции описывается формулой dst(I)=src1(I)&src2(I) if mask(I)!=0.

Функция

CVAPI(void) cvMinMaxLoc( const CvArr* arr, double* min_val, double* max_val,
CvPoint* min_loc CV_DEFAULT(NULL),
CvPoint* max_loc CV_DEFAULT(NULL),
const CvArr* mask CV_DEFAULT(NULL) );

определяет минимальное и максимальное значения массива, а также их местоположение,
где arr – массив(изображение) для поиска (1-канальный или многоканальный с установленным COI);
min_val – указатель на переменную для сохранения минимального значения;
max_val – указатель на переменную для сохранения максимального значения;
min_loc – указатель на точку местоположения минимума;
max_loc – указатель на точку местоположения максимума;
mask – маска для выбора подмассива.

Функция

CVAPI(void) cvMoveWindow( const char* name, int x, int y );

перемещает окно,
где name – название окна;
x, y – новые координаты верхнего левого угла окна.

Функция

CVAPI(void) cvResizeWindow( const char* name, int width, int height );

изменяет размер окна,
где name – название окна;
width, height – новые ширина и высота окна.

### 3.9. Поиск объекта по цвету в цветовом пространстве HSV

Для поиска объекта по цвету использовать модель HSV намного удобнее, т. к. координаты этой цветовой модели выбраны с учетом человеческого восприятия. **HSV** (англ. Hue, Saturation, Value – тон, насыщенность, значение) – цветовая модель, в которой координатами цвета являются [1–4]:
* **Hue** – цветовой тон (например, красный, зелёный или сине-голубой), варьируется в пределах 0–360°, однако иногда приводится к диапазону 0–100 или 0–1);
* **Saturation** – насыщенность, варьируется в пределах 0–100 или 0–1 (чем больше этот параметр, тем «чище» цвет, поэтому этот параметр иногда называют чистотой цвета, а чем ближе этот параметр к нулю, тем ближе цвет к нейтральному серому); * **Value** (значение цвета) или Brightness – яркость (также задаётся в пределах 0–100 и 0–1). Эта модель очень удобна для поиска на изображении объектов по цвету и яркости, например, пятно от лазерной указки, цветовые маркеры или просто объекты с выделяющимся цветом.

Преобразовать изображение из RGB в HSV очень просто:

```
// создаём изображения
IplImage* hsv = cvCreateImage( cvGetSize(src), 8, 3 );
IplImage* h_plane = cvCreateImage( cvGetSize(src), 8, 1 );
IplImage* s_plane = cvCreateImage( cvGetSize(src), 8, 1 );
IplImage* v_plane = cvCreateImage( cvGetSize(src), 8, 1 );
//  конвертируем в HSV
cvCvtColor( src, hsv, CV_BGR2HSV );
// разбиваем на каналы
cvCvtPixToPlane( hsv, h_plane, s_plane, v_plane, 0 );
```

То есть сначала просто создаются 4 изображения: одно для хранения изображения в формате HSV и три для последующего разделения изображения на отдельные каналы H, S и V.

Ниже приводится код утилиты, которая может быть полезна для определения минимальных и максимальных значений каналов HSV для выделения объекта на изображении. Программа преобразует изображение в формат HSV и затем, используя ползунки и функцию cvInRangeS(), формирует в дополнительных окнах «H range» бинарное изображение, которое получается из тех пикселей канала, значение которых больше Hmin и меньше Hmax. Далее для всех трёх получившихся пороговых изображений выполняется операция логического И (т. е. в результирующем окне «hsv and» белыми будут только те пиксели, которые есть у всех трёх пороговых изображений).

```
// конвертирует изображение в HSV
// и позволяет подобрать параметры
// Hmin, Hmax, Smin, Smax, Vmin, Vmax
// для выделения нужного объекта
```

```
//
// robocraft.ru
//
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* dst = 0;

// для хранения каналов HSV
IplImage* hsv = 0;
IplImage* h_plane = 0;
IplImage* s_plane = 0;
IplImage* v_plane = 0;
// для хранения каналов HSV после преобразования
IplImage* h_range = 0;
IplImage* s_range = 0;
IplImage* v_range = 0;
// для хранения суммарной картинки
IplImage* hsv_and = 0;

int Hmin = 0;
int Hmax = 256;

int Smin = 0;
int Smax = 256;

int Vmin = 0;
int Vmax = 256;

int HSVmax = 256;

//
// функции-обработчики ползунков
//
void myTrackbarHmin(int pos) {
	Hmin = pos;
	cvInRangeS(h_plane, cvScalar(Hmin), cvScalar(Hmax), h_range);
}

void myTrackbarHmax(int pos) {
	Hmax = pos;
```

```
	cvInRangeS(h_plane, cvScalar(Hmin), cvScalar(Hmax), h_range);
}

void myTrackbarSmin(int pos) {
	Smin = pos;
	cvInRangeS(s_plane, cvScalar(Smin), cvScalar(Smax), s_range);
}

void myTrackbarSmax(int pos) {
	Smax = pos;
	cvInRangeS(s_plane, cvScalar(Smin), cvScalar(Smax), s_range);
}

void myTrackbarVmin(int pos) {
	Vmin = pos;
	cvInRangeS(v_plane, cvScalar(Vmin), cvScalar(Vmax), v_range);
}

void myTrackbarVmax(int pos) {
	Vmax = pos;
	cvInRangeS(v_plane, cvScalar(Vmin), cvScalar(Vmax), v_range);
}

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// создаём картинки
	hsv = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 3 );
	h_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
	s_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
	v_plane = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
	h_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
	s_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
	v_range = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
	hsv_and = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1 );
	//  конвертируем в HSV
	cvCvtColor( image, hsv, CV_BGR2HSV );
```

```
// разбиваем на отдельные каналы
cvCvtPixToPlane( hsv, h_plane, s_plane, v_plane, 0 );

//
// определяем минимальное и максимальное значение
// у каналов HSV
double framemin=0;
double framemax=0;

cvMinMaxLoc(h_plane, &framemin, &framemax);
printf("[H] %f x %f\n", framemin, framemax );
Hmin = framemin;
Hmax = framemax;
cvMinMaxLoc(s_plane, &framemin, &framemax);
printf("[S] %f x %f\n", framemin, framemax );
Smin = framemin;
Smax = framemax;
cvMinMaxLoc(v_plane, &framemin, &framemax);
printf("[V] %f x %f\n", framemin, framemax );
Vmin = framemin;
Vmax = framemax;

// окна для отображения картинки
cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
cvNamedWindow("H",CV_WINDOW_AUTOSIZE);
cvNamedWindow("S",CV_WINDOW_AUTOSIZE);
cvNamedWindow("V",CV_WINDOW_AUTOSIZE);
cvNamedWindow("H range",CV_WINDOW_AUTOSIZE);
cvNamedWindow("S range",CV_WINDOW_AUTOSIZE);
cvNamedWindow("V range",CV_WINDOW_AUTOSIZE);
cvNamedWindow("hsv and",CV_WINDOW_AUTOSIZE);

cvCreateTrackbar("Hmin", "H range", &Hmin, HSVmax, myTrackbarHmin);
cvCreateTrackbar("Hmax", "H range", &Hmax, HSVmax, myTrackbarHmax);
cvCreateTrackbar("Smin", "S range", &Smin, HSVmax, myTrackbarSmin);
cvCreateTrackbar("Smax", "S range", &Smax, HSVmax, myTrackbarSmax);
cvCreateTrackbar("Vmin", "V range", &Vmin, HSVmax, myTrackbarVmin);
cvCreateTrackbar("Vmax", "V range", &Vmax, HSVmax, myTrackbarVmax);

//
// разместим окна по рабочему столу
//
if(image->width <1920/4 && image->height<1080/2){
      cvMoveWindow("original", 0, 0);
```

```
        cvMoveWindow("H", image->width+10, 0);
        cvMoveWindow("S", (image->width+10)*2, 0);
        cvMoveWindow("V", (image->width+10)*3, 0);
        cvMoveWindow("hsv and", 0, image->height+30);
        cvMoveWindow("H range", image->width+10, image->height+30);
        cvMoveWindow("S range", (image->width+10)*2, image->height+30);
        cvMoveWindow("V range", (image->width+10)*3, image->height+30);
}

while(true){

        // показываем картинку
        cvShowImage("original",image);

        cvShowImage( "H", h_plane );
        cvShowImage( "S", s_plane );
        cvShowImage( "V", v_plane );

        cvShowImage( "H range", h_range );
        cvShowImage( "S range", s_range );
        cvShowImage( "V range", v_range );

        // складываем
        cvAnd(h_range, s_range, hsv_and);
        cvAnd(hsv_and, v_range, hsv_and);

        cvShowImage( "hsv and", hsv_and );

        char c = cvWaitKey(33);
        if (c == 27) { // если нажата ESC – выходим
                break;
        }
}
printf("\n[i] Results:\n" );
printf("[H] %d x %d\n", Hmin, Hmax );
printf("[S] %d x %d\n", Smin, Smax );
printf("[V] %d x %d\n", Vmin, Vmax );

// освобождаем ресурсы
cvReleaseImage(&image);
cvReleaseImage(&hsv);
cvReleaseImage(&h_plane);
cvReleaseImage(&s_plane);
cvReleaseImage(&v_plane);
```

```
    cvReleaseImage(&h_range);
    cvReleaseImage(&s_range);
    cvReleaseImage(&v_range);
    cvReleaseImage(&hsv_and);
    // удаляем окна
    cvDestroyAllWindows();
    return 0;
}
```

Функция
CVAPI(void)  cvCvtColor( const CvArr* src, CvArr* dst, int code );
конвертирует изображение из одного цветового пространства в другое,
где src – исходное изображение;
dst – получаемое изображение;
code – параметр, указывающий тип конвертации:

```
#define  CV_BGR2BGRA    0
#define  CV_RGB2RGBA    CV_BGR2BGRA

#define  CV_BGRA2BGR    1
#define  CV_RGBA2RGB    CV_BGRA2BGR

#define  CV_BGR2RGBA    2
#define  CV_RGB2BGRA    CV_BGR2RGBA

#define  CV_RGBA2BGR    3
#define  CV_BGRA2RGB    CV_RGBA2BGR

#define  CV_BGR2RGB     4
#define  CV_RGB2BGR     CV_BGR2RGB

#define  CV_BGRA2RGBA   5
#define  CV_RGBA2BGRA   CV_BGRA2RGBA

#define  CV_BGR2GRAY    6
#define  CV_RGB2GRAY    7
#define  CV_GRAY2BGR    8
#define  CV_GRAY2RGB    CV_GRAY2BGR
#define  CV_GRAY2BGRA   9
#define  CV_GRAY2RGBA   CV_GRAY2BGRA
#define  CV_BGRA2GRAY   10
#define  CV_RGBA2GRAY   11

#define  CV_BGR2BGR565  12
#define  CV_RGB2BGR565  13
```

```
#define  CV_BGR5652BGR  14
#define  CV_BGR5652RGB  15
#define  CV_BGRA2BGR565 16
#define  CV_RGBA2BGR565 17
#define  CV_BGR5652BGRA 18
#define  CV_BGR5652RGBA 19

#define  CV_GRAY2BGR565 20
#define  CV_BGR5652GRAY 21

#define  CV_BGR2BGR555  22
#define  CV_RGB2BGR555  23
#define  CV_BGR5552BGR  24
#define  CV_BGR5552RGB  25
#define  CV_BGRA2BGR555 26
#define  CV_RGBA2BGR555 27
#define  CV_BGR5552BGRA 28
#define  CV_BGR5552RGBA 29

#define  CV_GRAY2BGR555 30
#define  CV_BGR5552GRAY 31

#define  CV_BGR2XYZ     32
#define  CV_RGB2XYZ     33
#define  CV_XYZ2BGR     34
#define  CV_XYZ2RGB     35

#define  CV_BGR2YCrCb   36
#define  CV_RGB2YCrCb   37
#define  CV_YCrCb2BGR   38
#define  CV_YCrCb2RGB   39

#define  CV_BGR2HSV     40
#define  CV_RGB2HSV     41

#define  CV_BGR2Lab     44
#define  CV_RGB2Lab     45

#define  CV_BayerBG2BGR 46
#define  CV_BayerGB2BGR 47
#define  CV_BayerRG2BGR 48
#define  CV_BayerGR2BGR 49

#define  CV_BayerBG2RGB CV_BayerRG2BGR
```

```
#define  CV_BayerGB2RGB CV_BayerGR2BGR
#define  CV_BayerRG2RGB CV_BayerBG2BGR
#define  CV_BayerGR2RGB CV_BayerGB2BGR

#define  CV_BGR2Luv     50
#define  CV_RGB2Luv     51
#define  CV_BGR2HLS     52
#define  CV_RGB2HLS     53

#define  CV_HSV2BGR     54
#define  CV_HSV2RGB     55

#define  CV_Lab2BGR     56
#define  CV_Lab2RGB     57
#define  CV_Luv2BGR     58
#define  CV_Luv2RGB     59
#define  CV_HLS2BGR     60
#define  CV_HLS2RGB     61

#define  CV_COLORCVT_MAX  100
```

Пример программы детектора кожи на изображении.

```
// пример использования CvAdaptiveSkinDetector
//
// robocraft.ru
//

#include <cv.h>
#include <highgui.h>
#include <cvaux.h> // для CvAdaptiveSkinDetector

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
      IplImage* image=0, *dst=0, *mask=0;

      // имя картинки задаётся первым параметром
      char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
      // получаем картинку
      image = cvLoadImage(filename, 1);
```

```
        printf("[i] image: %s\n", filename);
        assert( image != 0 );

        // покажем изображение
        cvNamedWindow( "original");
        cvShowImage( "original", image );

        // картинка для хранения изображения
        dst = cvCloneImage(image);

        // детектор кожи
        CvAdaptiveSkinDetector filter(1,
CvAdaptiveSkinDetector::MORPHING_METHOD_ERODE); //
MORPHING_METHOD_ERODE_DILATE

        // картинка для хранения результата (маски)
        mask = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1);

        // обработка
        filter.process(image, mask);

        // покажем маску
        cvNamedWindow( "mask");
        cvShowImage( "mask", mask);

        // пробегаемся по всем пикселям изображения
        // и увеличиваем у пикселей картинки, где установлена маска,
        // зелёную компоненту до максимума
        for( int y=0; y<dst->height; y++ ) {
                uchar* ptr = (uchar*) (dst->imageData + y * dst->widthStep); // изображе-
ние
                uchar* ptrM = (uchar*) (mask->imageData + y * mask->widthStep); //
маска
                for( int x=0; x<dst->width; x++ ) {
                        if(ptrM[x]){ // маска установлена
                                // 3 канала
                                //ptr[3*x] = ; // B
                                ptr[3*x+1] = 255; // G
                                //ptr[3*x+2] = ; // R
                        }
                }
        }

        // покажем картинку
```

```
    cvNamedWindow( "dst");
    cvShowImage( "dst", dst );

    // ждём нажатия клавиши
    cvWaitKey(0);

    // освобождаем ресурсы
    cvReleaseImage(& image);
    cvReleaseImage(&dst);
    cvReleaseImage(&mask);

    // удаляем окна
    cvDestroyAllWindows();
    return 0;
}
```

Для определения координат полученного объекта необходимо найти **центр масс** фигуры. Для этого обходим все пиксели картинки и для ненулевых пикселей складываем координаты и подсчитываем их количество.

Рассмотрим пример.

```
// находим центр масс
    //

    int Xc = 0;
    int Yc = 0;
    int counter = 0; // счётчик числа белых пикселей

    // пробегаемся по пикселям изображения
    for(y=0; y<result->height; y++)
    {
        uchar* ptr = (uchar*) (result->imageData + y * result->widthStep);
        for(x=0; x<result->width; x++)
        {
            if( ptr[x]>0 )
            {
                Xc += x;
                Yc += y;
                counter++;
            }
        }
    }

    if(counter!=0)
```

```
{
    center.x = float(Xc)/counter;
    center.y = float(Yc)/counter;
}
```

### 3.10. Свёртка

В случае работы с изображениями свёртка – это операция вычисления нового значения заданного пикселя, при которой учитываются значения окружающих его соседних пикселей. Главным элементом свёртки является так называемое **ядро свёртки** – это матрица произвольного размера и отношения сторон (чаще всего используется квадратная матрица (по умолчанию, размеры 3×3)) [1–4].

У ядра свёртки есть важный параметр – «**якорь**» – это элемент матрицы (чаще всего – центр), который прикладывается к заданному пикселю изображения.

Работает свёртка следующим образом. При вычислении нового значения выбранного пикселя изображения ядро свёртки прикладывается центром к этому пикселю. Соседние пиксели также накрываются ядром. Далее вычисляется сумма произведений значений пикселей изображения на значения, накрывшего данный пиксель элемента ядра. Полученная сумма и является новым значением выбранного пикселя. Теперь если применить свёртку к каждому пикселю изображения, то получится некий эффект, зависящий от выбранного ядра свертки.

Рассмотрим пример. Пусть есть клеточное поле, которое соответствует пикселям исходного изображния:

[47][48][49][ ][ ][ ][ ][ ][ ][ ][ ]
[47][50][42][ ][ ][ ][ ][ ][ ][ ][ ]
[47][48][42][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]

и есть ядро свёртки, представляющее собой матрицу 3×3 с якорем в центре и элементами:

[0][1][0]
[0][0][0]
[0][0][0].

Результат операции наложения ядра на пиксель со значением **50**:

[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][48][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ].

Результат = 47·0 + 48·1 + 49·0 + 47·0 + 50·0 + 42·0 + 47·0 + 48·0 + 42·0 = 48.

В OpenCV операция свёртки реализуется функцией **cvFilter2D**()

```
CVAPI(void) cvFilter2D( const CvArr* src, CvArr* dst, const CvMat* kernel,
                        CvPoint anchor CV_DEFAULT(cvPoint(-1,-1)));
```
где src – исходное изображение;
dst – изображение для сохранения результа;
kernel – ядро свёртки (массив из одного канала из элементов типа float);
anchor – якорь ядра ( (-1,-1) говорит, что якорь в центре (для ядра нечётной размерности)).

**Пример 3**

```
// применяем фильтр к изображению при помощи cvFilter2D()
//
// robocraft.ru
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* dst = 0;

int main(int argc, char* argv[])
{
    // имя картинки задаётся первым параметром
    char* filename = argc == 2 ? argv[1] : "Image0.jpg";
    // получаем картинку
    image = cvLoadImage(filename,1);

    printf("[i] image: %s\n", filename);
    assert( image != 0 );

    // клонируем картинку
    dst = cvCloneImage(image);

    // окно для отображения картинки
    cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("cvFilter2D",CV_WINDOW_AUTOSIZE);

    float kernel[9];
    kernel[0]=0;
    kernel[1]=0;
    kernel[2]=0;
```

```
        kernel[3]=0;
        kernel[4]=1;
        kernel[5]=0;

        kernel[6]=0;
        kernel[7]=0;
        kernel[8]=0;

        // матрица
        CvMat kernel_matrix=cvMat(3,3,CV_32FC1,kernel);

        // накладываем фильтр
        cvFilter2D(image, dst, &kernel_matrix, cvPoint(-1,-1));

        // показываем картинку
        cvShowImage("original",image);
        cvShowImage("cvFilter2D",dst);

        // ждём нажатия клавиши
        cvWaitKey(0);

        // освобождаем ресурсы
        cvReleaseImage(&image);
        cvReleaseImage(&dst);
        // удаляем окна
        cvDestroyAllWindows();
        return 0;
}
```

**Пример 4**

```
// сглаживание
        kernel[0]=0.1;
        kernel[1]=0.1;
        kernel[2]=0.1;

        kernel[3]=0.1;
        kernel[4]=0.1;
        kernel[5]=0.1;

        kernel[6]=0.1;
        kernel[7]=0.1;
        kernel[8]=0.1;
```

**Пример 5**

```
// увеличение чёткости
        kernel[0]=-0.1;
        kernel[1]=-0.1;
        kernel[2]=-0.1;

        kernel[3]=-0.1;
        kernel[4]=2;
        kernel[5]=-0.1;

        kernel[6]=-0.1;
        kernel[7]=-0.1;
        kernel[8]=-0.1;
```

**Пример 6**

```
// увеличение яркости
        kernel[0]=-0.1;
        kernel[1]=0.2;
        kernel[2]=-0.1;

        kernel[3]=0.2;
        kernel[4]=3;
        kernel[5]=0.2;

        kernel[6]=-0.1;
        kernel[7]=0.2;
        kernel[8]=-0.1;
```

**Пример 7**

```
// затемнение
        kernel[0]=-0.1;
        kernel[1]=0.1;
        kernel[2]=-0.1;

        kernel[3]=0.1;
        kernel[4]=0.5;
        kernel[5]=0.1;

        kernel[6]=-0.1;
        kernel[7]=0.1;
        kernel[8]=-0.1;
```

Таким образом, можно накладывать ядро с разными коэффициентами и получать различные эффекты. Свёртка – распространённая операция, лежащая в основе различных фильтров (размытие, повышение резкости, нахождение краёв). Но при использовании свёртки возникает вопрос, как должен вести себя алгоритм свёртки на краях изображения, в частности, в рассмотренном примере: как необходимо рассчитывать свёртку, если приложить якорь ядра к пикселю со значением **47**? Хорошим решением этой проблемы может быть создание изображение большего размера, чем исходное, у которого на краях будет заданное значение пикселей. Начинать обработку картинки необходимо тогда не с 0-го, а с 1-го пикселя и до (n–1)-го пискселя:

[0 ][0 ][0 ][0 ][0][0][0][0][0][0][0][0][0][0]
[0 ][47][48][49][ ][ ][ ][ ][ ][ ][ ][ ][ ][0]
[0 ][47][50][42][ ][ ][ ][ ][ ][ ][ ][ ][ ][0]
[0 ][47][48][42][ ][ ][ ][ ][ ][ ][ ][ ][ ][0]
[0 ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][0]
[0 ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][0]
[0 ][0 ][0 ][0 ][0][0][0][0][0][0][0][0][0][0].

Функция

```
CVAPI(void) cvCopyMakeBorder( const CvArr* src, CvArr* dst, CvPoint offset,
                              int bordertype, CvScalar value CV_DEFAULT(cvScalarAll(0)));
```

копирует исходное изображение, окружая его границей,
где src – исходное изображение;
dst – изображение для сохранения результа;
offset – смещение (координаты левого верхнего угла целевого изображения);
bordertype – тип границы:

```
#define IPL_BORDER_CONSTANT   0 // все пиксели границы заливаются value
#define IPL_BORDER_REPLICATE  1 // крайние пиксели изображения используются для заливки
// другие типы границы пока не поддерживаются:
#define IPL_BORDER_REFLECT    2 // пока не поддерживается
#define IPL_BORDER_WRAP       3 // пока не поддерживается
```

value – цвет заливки границы.

Пример использования:

```
// пример использования cvCopyMakeBorder()
//
// robocraft.ru
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>
```

```
IplImage* image = 0;
IplImage* dst = 0;
IplImage* dst2 = 0;

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc == 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename,1);
	// создаём картинки
	dst = cvCreateImage( cvSize(image->width+20, image->height+20), image-
>depth, image->nChannels);
	dst2 = cvCreateImage( cvSize(image->width+20, image->height+20), image-
>depth, image->nChannels);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// окно для отображения картинки
	cvNamedWindow("original",CV_WINDOW_AUTOSIZE);

cvNamedWindow("IPL_BORDER_CONSTANT",CV_WINDOW_AUTOSIZE);

cvNamedWindow("IPL_BORDER_REPLICATE",CV_WINDOW_AUTOSIZE);

	// обрамляем границей
	cvCopyMakeBorder(image, dst, cvPoint(10,10), IPL_BORDER_CONSTANT,
cvScalar(250));
	cvCopyMakeBorder(image, dst2, cvPoint(10,10), IPL_BORDER_REPLICATE,
cvScalar(250));


	// показываем картинку
	cvShowImage("original",image);
	cvShowImage("IPL_BORDER_CONSTANT",dst);
	cvShowImage("IPL_BORDER_REPLICATE",dst2);

	// ждём нажатия клавиши
	cvWaitKey(0);

	// освобождаем ресурсы
	cvReleaseImage(&image);
	cvReleaseImage(&dst);
```

```
    cvReleaseImage(&dst2);
    // удаляем окна
    cvDestroyAllWindows();
    return 0;
}
```

Функция свёртки cvFilter2D() внутри себя уже вызывает функцию cvCopyMakeBorder() с параметром **IPL_BORDER_REPLICATE**.

### 3.11. Операторы Собеля и Лапласа

Одна из важнейших свёрток – это вычисление производных. В математике и физике производные играют очень важную роль, то же самое можно сказать и про компьютерное зрение. Производная (функции в точке) – это скорость изменения функции (в данной точке). Определяется как предел отношения приращения функции к приращению ее аргумента при стремлении приращения аргумента к нулю. В случае изображения производная – это отношение значения приращения пикселя по y к значению приращению пикселя по x: dI = dy/dx [1–4].

Работая с изображением I, мы работаем с функцией двух переменных I(x,y), т. е. со скалярным полем. Поэтому более правильно говорить не о производной, а о градиенте изображения.

Градиент (от лат. gradiens – шагающий, растущий) – вектор, показывающий направление наискорейшего возрастания некоторой величины, значение которой меняется от одной точки пространства к другой (скалярного поля).

Если каждой точке M области многомерного пространства поставлено в соответствие некоторое (обычно действительное) число u, то говорят, что в этой области задано скалярное поле.

Градиент для каждой точки изображения (функция яркости) – 2-мерный вектор, компонентами которого являются производные яркости изображения по горизонтали и вертикали: grad I(x,y) = (dI/dx, dI/dy).

В каждой точке изображения градиентный вектор ориентирован в направлении наибольшего увеличения яркости, а его длина соответствует величине изменения яркости.

Вектор (в заданной точке) задаётся двумя значениями: длиной и направлением. Длина: sqrt( dx^2 + dy^2 ); направление – угол между вектором и осью x: atan(dy/dx).

Для дифференцирования изображения используется так называемый оператор Собеля. Это дискретный дифференциальный оператор, вычисляющий приближение градиента яркости изображения. Оператор вычисляет градиент яркости изображения в каждой точке. Так находится направление наибольшего увеличения яркости и величина её изменения в этом направлении. Результат показывает, насколько «резко» или «плавно» меняется яркость изображения в каждой точке, а значит, вероятность нахождения точки на грани, а также ориентацию границы. Таким образом, результатом работы оператора Со-

беля в точке области постоянной яркости будет нулевой вектор, а в точке, лежащей на границе областей различной яркости, – вектор, пересекающий границу в направлении увеличения яркости. Наиболее часто оператор Собеля применяется в алгоритмах выделения границ.

Оператор Собеля основан на свёртке изображения небольшими целочисленными фильтрами в вертикальном и горизонтальном направлениях, поэтому его относительно легко вычислять. Оператор использует ядра 3×3, с которыми свёртывают исходное изображение для вычисления приближённых значений производных по горизонтали и по вертикали.

Функция

```
CVAPI(void) cvSobel( const CvArr* src, CvArr* dst,
                     int xorder, int yorder,
                     int aperture_size CV_DEFAULT(3));
```

обеспечивает вычисление производной изображения (градиента), используя оператор Собеля (aperture_size = 1,3,5,7) или Щарра (aperture_size = -1),
где src – исходное изображение;
dst – изображение для сохранения результа;
xorder – порядок производной по x (0,1 или 2);
yorder – порядок производной по y (одновременно нулевой может быть только либо xorder, либо yorder);
aperture_size – размер ядра оператора Собеля (1,3,5,7).

Для x (для y значение получается транспонированием):

```
-1 0 1
-2 0 2
-1 0 1
```

При aperture_size==-1 используется **оператор Щарра** (Scharr) для x (для y значение получается транспонированием):

```
-3  0 3
-10 0 10
-3  0 3
#define CV_SCHARR -1
#define CV_MAX_SOBEL_KSIZE 7
```

Чтобы избежать переполнения, изображение должно быть 16-битным (IPL_DEPTH_16S) при 8-битном исходном изображении. Для преобразования получившегося изображения в 8-битное можно использовать cvConvertScale() или ConvertScaleAbs().

Функция

```
CVAPI(void)  cvConvertScale( const CvArr* src, CvArr* dst,
                             double scale CV_DEFAULT(1),
                             double shift CV_DEFAULT(0) );
#define cvCvtScale cvConvertScale
#define cvScale  cvConvertScale
#define cvConvert( src, dst )  cvConvertScale( (src), (dst), 1, 0 )
```

обеспечивает изменение типа массива (с опциональным изменением масштаба) – линейную трансформацию каждого элемента исходного массива (у многоканального изображения каждый канал обрабатывается отдельно) по формуле dst(x,y,c) = scale*src(x,y,c)+shift (функцию можно использовать для изменения типа изображения),
где src – исходное изображение;
dst – изображение для сохранения результа;
scale – масштаб;
shift – сдвиг – величина, добавляемая к каждому элементу.

Функция

```
CVAPI(void)  cvConvertScaleAbs( const CvArr* src, CvArr* dst,
                        double scale CV_DEFAULT(1),
                        double shift CV_DEFAULT(0) );
#define cvCvtScaleAbs  cvConvertScaleAbs
```

выполняет линейное масштабное преобразование изображения dst(x,y,c) = abs(scale*src(x,y,c)+shift) (данная функция работает только с изображениями типа 8u (8-битные беззнаковые), в других случаях может быть использована функция cvConvertScale() + cvAbsDiffS()).

Пример, демонстрирующий работу оператора Собеля.

```
// пример работы оператора Собеля - cvSobel()
//
// robocraft.ru
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* dst = 0;
IplImage* dst2 = 0;

int xorder = 1;
int xorder_max = 2;

int yorder = 1;
int yorder_max = 2;

//
// функция-обработчик ползунка
// порядок производной по X
void myTrackbarXorder(int pos) {
```

```
	xorder = pos;
}

//
// функция-обработчик ползунка
// порядок производной по Y
void myTrackbarYorder(int pos) {
	yorder = pos;
}

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename, 1);
	// создаём картинки
	dst = cvCreateImage( cvSize(image->width, image->height), IPL_DEPTH_16S,
image->nChannels);
	dst2 = cvCreateImage( cvSize(image->width, image->height), image->depth,
image->nChannels);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// окно для отображения картинки
	cvNamedWindow("original", CV_WINDOW_AUTOSIZE);
	cvNamedWindow("sobel", CV_WINDOW_AUTOSIZE);


	int aperture = argc == 3 ? atoi(argv[2]) : 3;

	cvCreateTrackbar("xorder", "original", &xorder, xorder_max,
myTrackbarXorder);
	cvCreateTrackbar("yorder", "original", &yorder, yorder_max,
myTrackbarYorder);

	while(1){

		// проверяем, чтобы порядок производных по X и Y был отличен от 0
		if(xorder==0 && yorder==0){
			printf("[i] Error: bad params for cvSobel() !\n");
			cvZero(dst2);
		}
```

```
            else{
                // применяем оператор Собеля
                cvSobel(image, dst, xorder, yorder, aperture);
                // преобразуем изображение к 8-битному
                cvConvertScale(dst, dst2);
            }

            // показываем картинку
            cvShowImage("original", image);
            cvShowImage("sobel", dst2);

            char c = cvWaitKey(33);
            if (c == 27) { // если нажата ESC – выходим
                break;
            }
        }

        // освобождаем ресурсы
        cvReleaseImage(& image);
        cvReleaseImage(&dst);
        cvReleaseImage(&dst2);
        // удаляем окна
        cvDestroyAllWindows();
        return 0;
}
```

Оператор Собеля представляет собой неточное приближение градиента изображения, но он достаточно хорош для практического применения во многих задачах. Однако с увеличением градиентного угла у оператора Собеля, с ядром 3×3 растут неточности, которые можно компенсировать, если использовать оператор Щарра (просто представляет другую версию ядра 3×3).

Оператор Лапласа позволяет вычислить так называемый лапласиан изображения – суммирование производных второго порядка. OpenCV содержит для этого функцию **cvLaplace**().

Функция

```
CVAPI(void) cvLaplace( const CvArr* src, CvArr* dst,
                int aperture_size CV_DEFAULT(3) );
```

получает лапласиан изображения (d2/dx + d2/dy)I. Фактически это оператор Собеля с xorder = yorder = 2. Исходное изображение может быть формата 8u или 32f. Результирующее изображение может быть формата 16s или 32f.

Рассмотрим пример.

```
// пример работы оператора Лапласа cvLaplace()
//
```

```
// robocraft.ru
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* dst = 0;
IplImage* dst2 = 0;

int main(int argc, char* argv[])
{
	// имя картинки задаётся первым параметром
	char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
	// получаем картинку
	image = cvLoadImage(filename, 1);
	// создаём картинки
	dst = cvCreateImage( cvGetSize(image), IPL_DEPTH_16S, image-
>nChannels);
	dst2 = cvCreateImage( cvGetSize(image), image->depth, image->nChannels);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// окно для отображения картинки
	cvNamedWindow("original", CV_WINDOW_AUTOSIZE);
	cvNamedWindow("cvLaplace", CV_WINDOW_AUTOSIZE);

	int aperture = argc == 3 ? atoi(argv[2]) : 3;

	// применяем оператор Лапласа
	cvLaplace(image, dst, aperture);

	// преобразуем изображение к 8-битному
	cvConvertScale(dst, dst2);

	// показываем картинку
	cvShowImage("original", image);
	cvShowImage("cvLaplace", dst2);

	cvWaitKey(0);
```

```
    // освобождаем ресурсы
    cvReleaseImage(& image);
    cvReleaseImage(&dst);
    cvReleaseImage(&dst2);
    // удаляем окна
    cvDestroyAllWindows();
    return 0;
}
```

### 3.12. Обработка краёв

Края (границы или контурные линии) – это кривые на изображении, вдоль которых происходит резкое изменение яркости или других видов неоднородностей. Проще говоря, край – это резкий переход/изменение яркости. Причины возникновения краёв: изменение освещенности; изменение цвета; изменение глубины сцены (ориентации поверхности). Края отражают важные особенности изображения и поэтому преобразование изображения в набор кривых производится в целях выделения существенных характеристик изображения и сокращения объема информации для последующего анализа [1–4].

Самым популярным методом выделения границ является детектор границ Кенни. Шаги алгоритма Кенни:

- удаление шума и лишних деталей из изображения;
- вычисление градиента изображения;
- формирование тонких краёв (edge thinning);
- связывание краёв в контурные линии (edge linking).

Детектор использует фильтр на основе первой производной от гауссиана. Так как он восприимчив к шумам, лучше не применять данный метод на необработанных изображения. Сначала исходные изображения нужно свернуть с гауссовым фильтром.

Границы на изображении могут находиться в различных направлениях, поэтому алгоритм Кенни использует четыре фильтра для выявления горизонтальных, вертикальных и диагональных границ. Если воспользоваться оператором обнаружения границ (например оператором Собеля), то получается значение для первой производной в горизонтальном (Gy) и вертикальном (Gx) направлениях. Из этого градиента можно получить угол направления границы Q = arctan(Gx/Gy), который округляется до одной из четырех углов, представляющих вертикаль, горизонталь и две диагонали (например, 0, 45, 90 и 135°). Затем идет проверка того, достигает ли величина градиента локального максимума в соответствующем направлении. Например, для сетки 3×3: если угол направления градиента равен нулю, точка будет считаться границей, если её интенсивность больше, чем у точки выше и ниже; если угол направления градиента равен 90°, точка будет считаться границей, если её интенсивность больше, чем у точки слева и справа; если угол направления градиента равен 135°, точка будет считаться границей, если её интенсивность больше, чем у точек,

находящихся в верхнем левом и нижнем правом углах; если угол направления градиента равен 45°, точка будет считаться границей, если её интенсивность больше, чем у точек, находящихся в верхнем правом и нижнем левом углах. Таким образом, получается двоичное изображение, содержащее границы (так называемые «тонкие края»).

Функция

CVAPI(void) cvCanny( const CvArr* image, CvArr* edges, double threshold1, double threshold2, int aperture_size CV_DEFAULT(3));

обеспечивает выполнение алгоритма Canny для поиска границ,
где image – одноканальное изображение для обработки (градации серого);
edges – одноканальное изображение для хранения границ, найденных функцией;
threshold1 – порог минимума;
threshold2 – порог максимума;
aperture_size – размер для оператора Собеля.

Рассмотрим пример.

```
// пример работы детектора границ Кенни - cvCanny()
//
// robocraft.ru
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>

IplImage* image = 0;
IplImage* gray = 0;
IplImage* dst = 0;

int main(int argc, char* argv[])
{
    // имя картинки задаётся первым параметром
    char* filename = argc == 2 ? argv[1] : "Image0.jpg";
    // получаем картинку
    image = cvLoadImage(filename,1);

    printf("[i] image: %s\n", filename);
    assert( image != 0 );

    // создаём 1-канальные картинки
    gray = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1);
    dst = cvCreateImage( cvGetSize(image), IPL_DEPTH_8U, 1);
```

```
    // окно для отображения картинки
    cvNamedWindow("original",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("gray",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("cvCanny",CV_WINDOW_AUTOSIZE);

    // преобразуем в градации серого
    cvCvtColor(image, gray, CV_RGB2GRAY);

    // получаем границы
    cvCanny(gray, dst, 10, 100, 3);

    // показываем картинки
    cvShowImage("original",image);
    cvShowImage("gray",gray);
    cvShowImage("cvCanny", dst );

    // ждём нажатия клавиши
    cvWaitKey(0);

    // освобождаем ресурсы
    cvReleaseImage(&image);
    cvReleaseImage(&gray);
    cvReleaseImage(&dst);
    // удаляем окна
    cvDestroyAllWindows();
    return 0;
}
```

### 3.13. Преобразование Хафа

Преобразование Хафа (Hough Transform) – это метод для поиска линий, кругов и других простых форм на изображении. Преобразование Хафа предназначено для поиска объектов, принадлежащих определённому классу фигур, с использованием процедуры голосования. Процедура голосования применяется к пространству параметров, из которого и получаются объекты определённого класса фигур по локальному максимуму в так называемом накопительном пространстве (accumulator space), которое строится при вычислении трансформации Хафа. Преобразование Хафа основывается на представлении искомого объекта в виде параметрического уравнения. Параметры этого уравнения представляют фазовое пространство (так называемый аккумуляторный массив/пространство, пространство Хафа). Затем берётся двоичное изображение (например, результат работы детектора границ Кенни). Перебираются все точки границ и делается предположение, что точка принадлежит линии искомого объекта. Таким образом, для каждой точки изображения рассчитывается нужное уравнение и полу-

чаются необходимые параметры, которые сохраняются в пространстве Хафа. Финальным шагом является обход пространства Хафа и выбор максимальных значений, за которые «проголосовало» больше всего пикселей картинки, что и даёт нам параметры для уравнений искомого объекта [1–4].

В основе теории преобразования Хафа лежит утверждение, что любая точка двоичного изображения может быть частью некоторого набора возможных линий.

Формула линии:

$y = a \cdot x + b$ // в декартовых координатах;

$p = x \cdot \cos(f) + y \cdot \sin(f)$ // в полярных координатах.

Прямую на плоскости можно представить в виде формулы

$x \cdot \cos(f) + y \cdot \sin(f) = R$,

где R – длина перпендикуляра, опущенного на прямую из начала координат;

f – угол между перпендикуляром к прямой и осью OX.

Угол f находится в пределах от 0 до $2 \cdot PI$, R ограничено размерами входного изображения.

Через каждую точку (x, y) изображения можно провести несколько прямых с разными R и f, т. е. каждой точке (x, y) изображения соответствует набор точек в фазовом пространстве (R, f), образующий синусоиду. Также каждой точке (R0, f0) пространства (R, f) можно поставить в соответствие счётчик, соответствующий количеству точек (x, y), лежащих на прямой:

$x \cdot \cos(f0) + y \sin(f0) = R0$.

Теперь непрерывное фазовое пространство нужно перевести в дискретное, введя сетку на пространстве (R, f), одной ячейке которой соответствует набор прямых с близкими значениями R и f.

Пример реализации преобразования Хафа.

```
// Hough transform – преобразование Хафа
// нахождение линий
//
// долгий и нудный перебор, поэтому лучше тестировать
// на маленьких картинках :)
//
// http://robocraft.ru
//

#include <cv.h>
#include <highgui.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

//
// преобразование Хафа для нахождения линий
```

```
//
// код на Паскале: http://forum.graphicon.ru/viewtopic.php?p=20222#p20222
//
void houghLine(IplImage* original, float accuracy=0.1)
{
	assert(original!=0);

	IplImage *src=0, *rgb=0;
	IplImage *bin=0;
	IplImage *phase=0;

	src=cvCloneImage(original);

	// заведём цветную картинку
	rgb = cvCreateImage(cvGetSize(src), 8, 3);
	cvConvertImage(src, rgb, CV_GRAY2BGR);

	// бинарная картинка для контуров
	bin = cvCreateImage(cvGetSize(src), IPL_DEPTH_8U, 1);
	cvCanny(src, bin, 50, 200);

	cvNamedWindow( "bin", 1 );
	cvShowImage( "bin", bin );

	// максимальное расстояние от начала координат – это длина диагонали
	int RMax = cvRound( sqrt( (double)(src->width*src->width + src->height*src-
>height) ) );

	// картинка для хранения фазового пространства Хафа (r, f)
	// 0 < r < RMax
	// 0 < f < 2*PI
	phase = cvCreateImage(cvSize(RMax, 180), IPL_DEPTH_16U, 1);
	cvZero(phase);

	int x=0, y=0, r=0, f=0;
	float theta=0;

	// рассматриваем пиксели изображения контуров
	for(y=0; y<bin->height; y++){
		uchar* ptr = (uchar*) (bin->imageData + y * bin->widthStep);
		for(x=0; x<bin->width; x++){
			if(ptr[x]>0){ // это пиксель контура?
				// рассмотрим все возможные прямые, которые могут
				// проходить через эту точку
```

```
                    for(f=0; f<180; f++){ //перебираем все возможные углы
наклона
                        short* ptrP = (short*) (phase->imageData + f * phase-
>widthStep);
                        for(r=0; r<RMax; r++){ // перебираем все возможные
расстояния от начала координат
                            theta=f*CV_PI/180.0; // переводим градусы в радиа-
ны

                            // Если решение уравнения достаточно хорошее
(точность больше заданной)
                            if ( abs(( (y)*sin(theta) + (x)*cos(theta)) - r) < accuracy
){
                                ptrP[r]++; // увеличиваем счётчик для этой точ-
ки фазового пространства.
                            }
                        }
                    }
                }
            }
        }

        cvNamedWindow( "phase", 1 );
        cvShowImage( "phase", phase );

        // увеличим фазовую картинку
        IplImage* phaseImage = cvCreateImage( cvSize(phase->width*3, phase-
>height*3), IPL_DEPTH_16U, 1);
        cvResize(phase, phaseImage);

        cvNamedWindow( "phaseImage", 1 );
        cvShowImage( "phaseImage", phaseImage);

        // Выбираем точку фазового пространства, которая набрала наибольшее
число попаданий
        unsigned int MaxPhaseValue = 0;
        float Theta=0;
        int R=0;
        for(f=0; f<180; f++){ //перебираем все возможные углы наклона
            short* ptrP = (short*) (phase->imageData + f * phase->widthStep);
            for(r=0; r<RMax; r++){ // перебираем все возможные расстояния от
начала координат
                if(ptrP[r]>MaxPhaseValue){
                    MaxPhaseValue = ptrP[r];
```

```
                    Theta = f;
                    R = r;
                }
            }
        }

#if 1
        printf("[M] %d\n", MaxPhaseValue);

        // нормировка
        float scaler = 0xFFFFFFFF/(float)MaxPhaseValue;
        for(y=0; y<phaseImage->height; y++){
            short* ptr = (short*) (phaseImage->imageData + y * phaseImage-
>widthStep);
            for(x=0; x<phaseImage->width; x++){
                ptr[x]*=scaler;
            }
        }
        cvNamedWindow( "phaseImage2", 1 );
        cvShowImage( "phaseImage2", phaseImage);
#endif

        // Рисуем линию по точкам, которые получили в результате преобразова-
ния
        Theta=Theta*CV_PI/180.0;
        for(y=0; y<rgb->height; y++){
            uchar* ptr = (uchar*) (rgb->imageData + y * rgb->widthStep);
            for(x=0; x<rgb->width; x++){
                if ( cvRound(((y) * sin(Theta) + (x) * cos(Theta))) == R){
                    ptr[3*x]=0;
                    ptr[3*x+1]=0;
                    ptr[3*x+2]=255;
                }
            }
        }

        cvNamedWindow( "line", 1 );
        cvShowImage( "line", rgb );

        // освобождаем ресурсы
        cvReleaseImage(&src);
        cvReleaseImage(&rgb);

        cvReleaseImage(&bin);
```

```
    cvReleaseImage(&phase);
    cvReleaseImage(&phaseImage);
}

int main(int argc, char* argv[])
{
    IplImage *original=0;

    // имя картинки задаётся первым параметром
    char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
    // получаем картинку
    original = cvLoadImage(filename, 0);

    printf("[i] image: %s\n", filename);
    assert( original != 0 );

    // покажем изображения
    cvNamedWindow( "original", 1 );
    cvShowImage( "original", original );

    houghLine(original);

    cvWaitKey(0);

    cvReleaseImage(&original);

    // удаляем окна
    cvDestroyAllWindows();
    return 0;
}
```

Кроме того, в OpenCV уже есть функция, реализующая преобразование Хафа для поиска линий.

Функция

```
CVAPI(CvSeq*)  cvHoughLines2( CvArr* image, void* line_storage, int method,
                              double rho, double theta, int threshold,
                              double param1 CV_DEFAULT(0), double param2
CV_DEFAULT(0));
```

обеспечивает нахождение линий на двоичном изображении, используя преобразование Хафа,

где image – 8-битное двоичное изображение (в случае вероятностного метода изображение будет модифицироваться);

line_storage – хранилище памяти для сохранения найденных линий (можно использовать матрицу 1хЧисло_линий CvMat);

method – метод:
#define CV_HOUGH_STANDARD 0
#define CV_HOUGH_PROBABILISTIC 1
#define CV_HOUGH_MULTI_SCALE 2
**CV_HOUGH_STANDARD** – классический вариант трансформации Хафа, где каждая линия представляется двумя числами типа float (rho, theta), rho – дистанция между точкой (0,0) и линией, а theta – угол между осью x и нормалью к линии (т. е. матрица должна иметь тип of CV_32FC2);
**CV_HOUGH_PROBABILISTIC** – вероятностный метод трансформации Хафа (более эффективен в случае изображений с несколькими длинными линейными сегментами), возвращает сегменты линии, которые представляются точками начала и конца (т. е. матрица должна иметь тип of CV_32SC4);
**CV_HOUGH_MULTI_SCALE** – масштабный вариант классической трансформации Хафа (линии представлены так же, как и в CV_HOUGH_STANDARD);
rho – разрешение по дистанции;
theta – разрешение по углу (в радианах);
threshold – пороговый параметр, линия возвращается, если аккумулирующий параметр больше порогового значения;
param1 – первый параметр (в зависимости от метода трансформации:
CV_HOUGH_STANDARD – 0 – не используется;
CV_HOUGH_PROBABILISTIC – минимальная длина линии;
CV_HOUGH_MULTI_SCALE – делитель разрешения по дистанции (rho/param1));
param2 – второй параметр (в зависимости от метода трансформации:
CV_HOUGH_STANDARD – 0 – не используется;
CV_HOUGH_PROBABILISTIC – максимальный промежуток между сегментами линии, лежащими на одной линии, чтобы считать их одним сегментом (объединить их вместе);
CV_HOUGH_MULTI_SCALE – делитель разрешения по углу (theta/param1)).

Если line_storage – указатель на хранилище памяти, то функция вернёт указатель на первый элемент последовательности, содержащей найденные линии.

Вероятностное преобразование Хафа (Probabilistic Hough Transform) заключается в том, что для нахождения объекта достаточно провести преобразование Хафа только для части (a) точек исходного изображения, 0 % <= a <= 100 %. То есть сначала провести выделение «контрольных» точек с изображения, затем для него провести преобразование Хафа.

Рассмотрим пример.

```
// нахождение линий на изображении,
// с использованием преобразования Хафа cvHoughLines2()
//
// модифицированный пример samples\c\houghlines.c
//
// http://robocraft.ru
//
```

```
#include <cv.h>
#include <highgui.h>
#include <math.h>

int main(int argc, char* argv[])
{
	IplImage* src = 0;
	IplImage* dst=0;
	IplImage* color_dst=0;

	// имя картинки задаётся первым параметром
	char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
	// получаем картинку (в градациях серого)
	src = cvLoadImage(filename, CV_LOAD_IMAGE_GRAYSCALE);

	if( !src ){
		printf("[!] Error: cant load image: %s \n", filename);
		return -1;
	}

	printf("[i] image: %s\n", filename);


	// хранилище памяти для хранения найденных линий
	CvMemStorage* storage = cvCreateMemStorage(0);
	CvSeq* lines = 0;
	int i = 0;


	dst = cvCreateImage( cvGetSize(src), 8, 1 );
	color_dst = cvCreateImage( cvGetSize(src), 8, 3 );

	// детектирование границ
	cvCanny( src, dst, 50, 200, 3 );

	// конвертируем в цветное изображение
	cvCvtColor( dst, color_dst, CV_GRAY2BGR );

	// нахождение линий
	lines = cvHoughLines2( dst, storage, CV_HOUGH_PROBABILISTIC, 1,
CV_PI/180, 50, 50, 10 );

	// нарисуем найденные линии
	for( i = 0; i < lines->total; i++ ){
```

```
            CvPoint* line = (CvPoint*)cvGetSeqElem(lines,i);
            cvLine( color_dst, line[0], line[1], CV_RGB(255,0,0), 3, CV_AA, 0 );
        }

        // показываем
        cvNamedWindow( "Source", 1 );
        cvShowImage( "Source", src );

        cvNamedWindow( "Hough", 1 );
        cvShowImage( "Hough", color_dst );

        // ждём нажатия клавиши
        cvWaitKey(0);

        // освобождаем ресурсы
        cvReleaseMemStorage(&storage);
        cvReleaseImage(&src);
        cvReleaseImage(&dst);
        cvReleaseImage(&color_dst);
        cvDestroyAllWindows();
        return 0;
}
```

Рассмотрим преобразование Хафа для нахождения окружностей (Hough Circle Transform). Точки окружности можно представить формулой
(x - a)^2 + (y - b)^2 = R2,
где (a, b) – координаты центра окружности, а R – её радиус.

То есть формула, задающая семейство окружностей, имеет вид
F (a, b, R, x, y) = (x - a)^2 + (y - b)^2 - R2.

Для нахождения окружностей нужно задавать уже 3 параметра – координаты центра окружности и её радиус. Это приводит к увеличению пространства Хафа на целое измерение, что в итоге сказывается на скорости работы. Поэтому для поиска окружностей применяется так называемый градиентный метод Хафа (Hough gradient method).

Эффективность использования преобразования Хафа резко падает при увеличении размерности фазового пространства, поэтому перед его применением желательно минимизировать каким-либо образом количество параметров кривой. Можно существенно снизить количество кривых, потенциально проходящих через данную точку изображения, если рассматривать только кривые, касательная которой перпендикулярна градиенту яркости изображения в рассматриваемой точке. Таким образом, можно, например, свести задачу выделения окружностей с неизвестным радиусом к 2-мерному фазовому пространству.

Алгоритм нахождения окружностей состоит из следующих шагов:

1) находятся границы на изображении;

2) для ненулевых точек высчитывается градиент (через вычисление 1-й производной по X и Y через cvSobel());

3) определяются центры окружностей;

4) относительно центра определяются ненулевые точки, лежащие на одном расстоянии.

Функция

```
CVAPI(CvSeq*) cvHoughCircles( CvArr* image, void* circle_storage,
                    int method, double dp, double min_dist,
                    double param1 CV_DEFAULT(100),
                    double param2 CV_DEFAULT(100),
                    int min_radius CV_DEFAULT(0),
                    int max_radius CV_DEFAULT(0));
```

обеспечивает нахождение кругов на изображении,
где image – исходное изображение (8-битное, 1-канальное, градации серого);
line_storage – хранилище памяти для сохранения найденных кругов (можно использовать матрицу типа CV_32FC3, 1xЧисло_кругов CvMat), круг хранится в виде 3 чисел типа float: координаты центра (x,y) и радиус;
method – метод, на настоящее время реализован только:
#define CV_HOUGH_GRADIENT 3
dp – разрешение сумматора, используемое для детектирования центров кругов (1 – одинаково с исходным изображением, 2 – половина высоты/ширины и т. д.);
min_dist – минимальная дистанция между центрами детектируемых кругов;
param1 – первый параметр (в зависимости от метода трансформации:
CV_HOUGH_GRADIENT – верхнее пороговое значение, передаваемое детектору границ Кенни (нижнее пороговое значение будет в 2 раза меньше);
param2 – второй параметр (в зависимости от метода трансформации:
CV_HOUGH_GRADIENT – суммирующее пороговое значение детектированя центров;
min_radius – минимальный радиус круга;
max_radius – максимальный радиус круга).

Рассмотрим пример.

```
// модифицированный пример example 6-1 Hough circles
// нахождение кругов на изображении,
// с использованием трансформации Хафа cvHoughCircles()
//
// из книги:
//   Learning OpenCV: Computer Vision with the OpenCV Library
//     by Gary Bradski and Adrian Kaehler
//     Published by O'Reilly Media, October 3, 2008
//
//
// http://robocraft.ru
//
```

```
#include <cv.h>
#include <highgui.h>
#include <math.h>

int main(int argc, char* argv[])
{
	IplImage* image = 0;
	// имя картинки задаётся первым параметром
	char* filename = argc >= 2 ? argv[1] : "Image0.jpg";
	// получаем картинку (в градациях серого)
	image = cvLoadImage(filename, CV_LOAD_IMAGE_GRAYSCALE);

	printf("[i] image: %s\n", filename);
	assert( image != 0 );

	// загрузим оригинальное изображение
	IplImage* src = cvLoadImage(filename );

	// хранилище памяти для кругов
	CvMemStorage* storage = cvCreateMemStorage(0);
	// сглаживаем изображение
	cvSmooth(image, image, CV_GAUSSIAN, 5, 5 );
	// поиск кругов
	CvSeq* results = cvHoughCircles(
		image,
		storage,
		CV_HOUGH_GRADIENT,
		10,
		image->width/5
		);
	// пробегаем по кругам и рисуем их на оригинальном изображении
	for( int i = 0; i < results->total; i++ ) {
		float* p = (float*) cvGetSeqElem( results, i );
		CvPoint pt = cvPoint( cvRound( p[0] ), cvRound( p[1] ) );
		cvCircle( src, pt, cvRound( p[2] ), CV_RGB(0xff,0,0) );
	}
	// показываем
	cvNamedWindow( "cvHoughCircles", 1 );
	cvShowImage( "cvHoughCircles", src);

	// ждём нажатия клавиши
	cvWaitKey(0);
```

```
    // освобождаем ресурсы
    cvReleaseMemStorage(&storage);
    cvReleaseImage(& image);
    cvReleaseImage(&src);
    cvDestroyAllWindows();
    return 0;
}
```

## 4. Указания по выполнению лабораторных работ

### 4.1. Цель лабораторных работ

Систематизация знаний и и закрепление навыков по разработке программ с использованием библиотеки OpenCV.

### 4.2. Описание лабораторных работ

Лабораторные работы включают 13 заданий, основанных на материалах и примерах, приведённых в разд. 3. Каждое задание предполагает написание программы по примеру, приведённому в соответствующем подразделе, её отладку и проверку работоспособности. Преподаватель может немного изменить исходные данные для разработки программы.

### 4.3. Предварительное задание к лабораторным работам

1. Изучить основы OpenCV, изложенные в разд. 1.
2. Изучить принципы реализации базовых функций, изложенные в разд. 2.

### 4.4. Порядок выполнения лабораторных работ

Порядок выполнения лабораторных работ (порядок выполнения заданий) определяет преподаватель.

### 4.5. Контрольные вопросы

1. Приведите структуру библиотеки OpenCV.
2. В чём отличие OpenCV версии 2.2 от версии 1.0?
3. Приведите порядок установки OpenCV.
4. Как организуется ввод-вывод данных в OpenCV?
5. Как осуществляются загрузка и вывод изображения?
6. В чём особенности загрузки и вывода видео?
7. Каким образом осуществляется подключение полосы прокрутки?
8. Как осуществляется захват видео?
9. Какие функции используются и как организуется запись видео в файл?
10. Каким образом обрабатываются события от мыши?
11. Что такое сглаживание изображения и какие функции его реализуют?
12. Как осуществляется изменение размеров изображения?
13. Что такое ROI? Какие функции используются для выделения ROI?
14. Дайте определение морфологическим преобразованиям и приведите функции, которые их реализуют.
15. Как произвести заливку изображения?

16. Что такое альфа-смешивание и с помощью каких функций можно его реализовать?

17. Для чего необходимо пороговое преобразование и какие функции его осуществляют?

18. Какие функции используются для поиска объектов по цвету?

19. В чём особенность поиска объектов по цвету в цветовом пространстве HSV?

20. Для чего необходима и как реализуется свёртка?

21. Что такое градиент, как и с помощью каких функций он вычисляется?

22. Как функционирует и с помощью каких функций реализуется детектор Кенни?

23. В чём назначение и сущность преобразования Хафа? Какие функции позволяют реализовать преобразование Хафа?

## Литература

1. OpenCV шаг за шагом [Электронный ресурс] – Режим доступа : http: //robocraft.ru/page/opencv/. – Дата доступа : 21.01.2015.

2. Gonzales, R. C. Digital image processing / R. C. Gonzales, R. E. Woods. – Prentice-Hall: 2nd edition, 2002. – 793 p.

3. Шапиро, Л. Компьютерное зрение / Л. Шапиро, Дж. Стокман ; пер. с англ. ; под ред. С. М. Соколова. – М. : БИНОМ. Лаборатория знаний, 2006. – 752 с.

4. Яне, Б. Цифровая обработка изображений / Б. Яне ; пер. с англ. ; под ред. А. М. Измайловой. – М. : ТЕХНОСФЕРА, 2007. – 584 с.

Св. план 2015, поз. 53

*Учебное издание*

**Волков** Кирилл Аркадьевич
**Цветков** Виктор Юрьевич
**Конопелько** Валерий Константинович

# *АЛГОРИТМЫ И ПРОГРАММНЫЕ БИБЛИОТЕКИ ОБРАБОТКИ МУЛЬТИМЕДИЙНОЙ ИНФОРМАЦИИ*

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *Е. Г. Бабичева*

Подписано в печать 25.05.2017. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 5,93. Уч.-изд. л. 5,7. Тираж 75 экз. Заказ 59.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6