

Александр Кручинин

Распознавание образов с использованием OpenCV

Материалы блога <http://recog.ru>

Версия документа 1.0

Материалы в данном файле собраны из блога «Распознавание образов для программистов», размещенного по адресу <http://recog.ru>. В большинстве случаев материал не обработан и просто скопирован для структурирования. Некоторые материалы, например про установку OpenCV на платформу Android, не вошли, поскольку либо написаны другими авторами, либо нуждаются в переработке. Но в будущем, если такой подход к размещению материала будет интересен, то будут новые версии документа. Прошу читателей не очень обижаться на автора, если где-то встречаются непонятные моменты, различаются стили написания — весь материал скопирован почти без изменения, а поскольку он был написан за несколько лет. Другим следствием является то, что материалы не являются полным руководством к библиотеке OpenCV, поскольку автором были рассмотрены только интересовавшие его вопросы. А написание полноценного руководства требует значительного времени. Часть материалов является переводом документации, часть программ носили экспериментальный характер и актуальны только для обучения.

Все материалы доступны на сайте <http://recog.ru>, старой версии блога <http://blog.vidikon.com> и на youtube — <http://www.youtube.com/user/VIDIKONCOM>, где представлены некоторые примеры.

При наличии пожеланий, замечаний, рекомендаций просьба писать по адресам support@vidikon.com, <http://recog.ru>.

Александр Кручинин

30 декабря 2011



1. ВВЕДЕНИЕ

1.1. Подключение OpenCV к программе

Также, как и любые другие библиотеки OpenCV можно подключить к программе, путем включения в проект `lib` и `h` файлов, например, для версий начиная с 2.2 необходимо подключать `h` и `lib` файлы следующего формата:

```
#include "opencv2/core/core_c.h"
#include "opencv2/imgproc/imgproc_c.h"
#include "opencv2/highgui/highgui_c.h"

#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
```

Библиотеки:

`opencv_core220d.lib`

`opencv_highgui220d.lib`

`opencv_imgproc220d.lib`

(естественно для Release должны быть релизные библиотеки без буквы "d" в конце).

В старых версиях использовались немного другие `h` файлы и `lib`. Например:

```
#include "cv.h"
#include "highgui_c.h"
```

Естественно и `lib` другие.

В дальнейшем в различных примеров будут использоваться разные версии `opencv`, но мы на этом не будем заострять внимание.

1.2. OpenCV 2.0: проблемы при установке в Windows

Вы установили библиотеку, но... обнаружили, что нет ни `.lib` файлов ни знакомых вам `.dll` файлов. Оказывается, чтобы добраться для них надо откомпилировать всю библиотеку, например так как показано здесь:

<http://mirror2image.wordpress.com/2009/10/20/switching-to-opencv-2-0-with-vs2005/>

Вкратце прокомментируем:



- 1) инсталляция OpenCV 2.0a;
- 2) инсталляция CMake (<http://www.cmake.org/files/v2.6/cmake-2.6.4-win32-x86.exe>);
- 3) перезагрузка компьютера;
- 4) запуск cmake-gui.exe;
- 5) выбираем места, где находится начальный и конечный (после компиляции) код;
- 6) нажимаем кнопку «Configure» и выбираем, например Visual Studio 2005 (естественно она у вас должна присутствовать);
- 7) нажимаем «Generate» — после этого должен быть сгенерирован проект;
- 8) запускаем проект OpenCV в Visual Studio 2005 и компилируем — ждём несколько минут результата;
- 9) ну всё — у вас есть dll и lib файлы, которые вы можете поместить куда хотите (а именно в те же папки lib и bin OpenCV 2.0).

Все пользователи, у которых есть лицензионные Visual Studio 6.0 или 2003, могут идти покупать новые версии, т.к. откомпилировать в них программу для OpenCV 2.0a вы не сможете.

1.3. Статическое подключение OpenCV 1.1 к проекту в VC 2003

Задача, конечно, врятли актуальная. Поскольку большинство пользуется OpenCV более новых версий и более новые студии. Однако, возникла такая ситуация. Необходимо создать проект на visual studio 2003 с использованием некоторых средств OpenCV, которая была бы статически подключена. Первое — это то, что версии OpenCV, начиная с 2.0 отпадают, поскольку они предназначены только для работы, начиная с vs 2005. Второе — динамически все работает нормально, однако когда речь заходит о статической линковке возникает ряд проблем. Ниже описано, какие действия надо сделать, что откомпилировать и подключить статически OpenCV к вашему проекту в vs 2003.

Компиляция OpenCV 1.1

1. Открыть в vs 2003 файл opencv.dsw, согласившись с преобразованиями в более новую версию, поскольку это файл проекта vs 6.0
 2. Сделать активным Release.
- Пункты 3-6 для всех 14 проектов
3. Установить Project->Properties->General->Configuration Type = Static Library(.lib) для всех проектов (их 14). Хотя, конечно cvsample и cvtest не нужны.
 4. Установить Project->Properties->C/C++->Code Generation->Runtime Library = /MT.
 5. Установить Project->Properties->C/C++->General->Debug Information Format = Disabled
 6. Установить Project->Properties->Librarian->General->Additional Dependences = msvcr71.lib
 7. Для проектов libjasper, libjpeg, libpng, libtiff, zlib поменять Project->Properties->Librarian->General->Output file на \$(OutDir)/Название.lib, поскольку там прописаны пути неправильно.



Компилируем. Ждем. Должно быть 0 ошибок. После чего копируем lib в папки lib и otherlibs_graphics\lib\.

Подключение

1. Открываем ваш проект в vs 2003, к которому необходимо подключить Opensv 2.1.
2. Установить Project->Properties->C/C++->Code Generation->Runtime Library = /MT.
3. Установить Project->Properties->C/C++->General->Debug Information Format = Disabled
4. Установить в Project->Properties->Linker->Input->Additional Dependences = libjpeg.lib libjasper.lib libpng.lib libtiff.lib cv.lib highgui.lib zlib.lib cxts.lib cvaux.lib ml.lib vfw32.lib videoInput.lib msvcr71.lib comctl32.lib. (Возможно там ещё нужны будут библиотеки в зависимости от вашего проекта).
5. Установить в Project->Properties->Linker->Input->Ignore Specific Library = atlthunk.lib;LIBCMT.lib

Компилируем. Должен получиться большой exe файл (в моем проекте 1.7MB), который должен запускаться без OpenCV. У меня работало на XP и на Seven.



2. HighGUI

Модуль HighGUI (High-level Graphical User Interface) предоставляет функции, которые позволяют взаимодействовать с операционной системой, файловой системой и аппаратными средствами ЭВМ такими, как камеры. HighGUI позволяет открывать окна, показывать изображения, читать и записывать графические файлы и видео, просто обращаться с мышью и клавиатурой.

2.1. Простой GUI

В таблице 2.1 представлены функции простого GUI для работы с окнами, клавиатурой и мышью.

Табл. 2.1. Функции простого GUI

Функция	Назначение
cvNamedWindow	Создание окна.
cvDestroyWindow	Уничтожение окна.
cvDestroyAllWindows	Уничтожение всех окон созданных с помощью HighGUI.
cvResizeWindow	Изменить размеры окна.
cvMoveWindow	Изменить позицию окна.
cvGetWindowHandle	Получить указатель на окно.
cvGetWindowName	Получить имя окна по указателю.
cvShowImage	Отобразить картинку в окно.
cvCreateTrackbar	Создать ползунок и добавить его к окну.
cvGetTrackbarPos	Узнать позицию ползунка.
cvSetTrackbarPos	Установить позицию ползунка.
cvSetMouseCallback	Установить функцию обработки события от мыши.
cvWaitKey	Ожидание нажатия клавиши на клавиатуре.

Опишем основные функции, которые вы будете использовать при проектировании и тестировании приложений.

cvNamedWindow

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

Параметры:



name

Название окна, которое будет использоваться для обращения к нему.

flags

Флаг окна. На настоящий момент поддерживается только CV_WINDOW_AUTOSIZE. Если он установлен, то окно автоматически подстраивается под загружаемое на него с помощью функции cvShowImage изображение.

Если окно с таким именем уже существует, то функция не делает ничего. Иначе создается окно, которое поддерживает место для картинки и для ползунка.

cvResizeWindow

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

Параметры:

name

Имя окна, размеры которого меняются.

width

Новая ширина.

height

Новая высота.

cvShowImage

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

Параметры:



name

Имя окна.

image

Указатель на изображение, которое должно быть отображено.

cvWaitKey

```
int cvWaitKey(
```

```
    int delay=0
```

```
);
```

Параметры:

delay

Задержка в миллисекундах.

Функция ожидает нажатия клавиши или пока не окончится задержка. Возвращает код нажатой клавиши или -1, если ничего не было нажато за время задержки.

2.2. HighGUI: Загрузка и сохранение изображений

Модуль HighGUI (High-Level Graphical User Interface) предоставляет функции, которые позволяют взаимодействовать с операционной системой, файловой системой и аппаратными средствами ЭВМ такими, как камеры. HighGUI позволяет открывать окна, показывать изображения, читать и записывать графические файлы и видео, просто обращаться с мышью и клавиатурой.

Для загрузки и сохранения изображений существуют всего две функции `cvLoadImage` и `cvSaveImage`.

cvLoadImage

```
IplImage* cvLoadImage(
```

```
    const char* filename,
```

```
    int flags=CV_LOAD_IMAGE_COLOR
```

```
);
```

Параметры:



filename

Имя файла, который необходимо загрузить.

flags

Определяет цвет и глубину загруженного изображения. Цвет определяет, должно ли загруженное изображение быть преобразовано на 3 канала (CV_LOAD_IMAGE_COLOR), 1 канал (CV_LOAD_IMAGE_GRAYSCALE), или оставлено без изменений так, как это было настроено во входном файле.

Глубина определяет, должно ли загруженное изображение будет преобразовано к 8-битному цветовому формату, как было общепринято в предыдущих версиях OpenCV или оставить так, как это было настроено во входном файле. Если выбрано CV_LOAD_IMAGE_ANYDEPTH то формат пикселей может быть 8-битный беззнаковый, 16-битный беззнаковый, 32-битный знаковый или 32 битный с плавающей точкой. Если Вы хотите загрузить изображение настолько близко к находящемуся в файле, насколько это возможно, то необходимо установить CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR.

Функция cvLoadImage загружает изображение из указанного файла и возвращает указатель на загруженное изображение. В настоящее время поддерживаются следующие форматы файлов:

Windows bitmaps - BMP, DIB;

JPEG files - JPEG, JPG, JPE;

Portable Network Graphics - PNG;

Portable image format - PBM, PGM, PPM, PXM, PNM;

Sun rasters - SR, RAS;

TIFF files - TIFF, TIF;

OpenEXR HDR images - EXR;

JPEG 2000 images - jp2.

cvSaveImage

```
int cvSaveImage(  
    const char* filename,  
    const CvArr* image  
);
```

Параметры:



filename

Имя файла, куда будет сохраняться изображение.

image

Указатель на изображение, которое должно быть сохранено.

Функция `cvSaveImage` сохраняет изображение в указанный файл. Формат изображения выбирается в зависимости от расширения файла указанного в параметре `filename`.

В листинге 2.1 приведен пример загрузки изображения, отображения его в графическом окне, ожидание нажатия клавиши и сохранение изображения в файле другого формата.

```
#include "cv.h"
#include "highgui.h"

int _tmain(int argc, _TCHAR* argv[])
{
    IplImage* img = cvLoadImage( "test.jpg"); //Загрузка изображения в
                                           //формате JPEG

    cvNamedWindow("Example1", CV_WINDOW_AUTOSIZE ); //Создание окна
    cvShowImage("Example1", img ); //Вывод изображения в окно
    cvWaitKey(0); //Ожидание нажатия любой клавиши
    cvSaveImage("test.png",img); //Сохранение изображения в формате PNG
    cvReleaseImage( &img ); //Удаление изображения из памяти
    cvDestroyWindow("Example1"); //Удаление окна

    return 0;
}
```

Листинг 2.1. Загрузка, вывод на экран и преобразование изображения

2.3. HighGUI: Видео ввод/вывод

Важным понятие при работе с графикой в OpenCV является структура `CvCapture`. Данная структура устанавливает связь с видео-потокком. В таблице 2.2 приведены функции для работы с видео. Далее опишем основные функции.



Табл.2.2. Функции для работы с видео

Функция	Назначение
cvCreateFileCapture	Создаёт и инициализирует структуру CvCapture для чтения видео-потока из файла.
cvCreateCameraCapture	Создаёт и инициализирует структуру CvCapture для чтения видео-потока с камеры.
cvReleaseCapture	Освобождает структуру CvCapture.
cvGrabFrame	Захватывает фрейм из видео-потока.
cvRetrieveFrame	Возвращает указатель на изображение захваченного фрейма.
cvQueryFrame	Захватывает фрейм, осуществляет декомпрессию и возвращает картинку.
cvGetCaptureProperty	Возвращает настройки камеры или характеристику видео-файла.
cvSetCaptureProperty	Устанавливает настройки камеры или характеристику видео-файла.
cvCreateVideoWriter	Создаёт структуру для видеозаписи.
cvReleaseVideoWriter	Заканчивает запись в видео-файл и освобождает структуру видеозаписи.
cvWriteFrame	Добавляет один фрейм к видео-файлу.

cvCreateFileCapture

```
CvCapture* cvCreateFileCapture(
    const char* filename
);
```

Параметры:

filename

Имя видеофайла.

Возвращает указатель на структуру CvCapture.

cvCreateCameraCapture

```
CvCapture* cvCreateCameraCapture(
    int index
);
```

Параметры:



index

Номер камеры для использования. Если камера только одна или неважно какой камерой пользоваться, то можно указать значение -1.

Возвращает указатель на структуру CvCapture. Только два интерфейса камер могут использоваться в Windows: Video for Windows (VFW) и Matrox Imaging Library (MIL); и два в Linux: V4L и FireWire (IEEE1394).

cvQueryFrame

```
IpplImage* cvQueryFrame(  
    CvCapture* capture  
);
```

Параметры:

capture

Структура видео-потока.

Функция захватывает фрейм, осуществляет декомпрессию и возвращает картинку. Она является комбинацией функций cvGrabFrame и cvRetrieveFrame за один вызов. Возвращаемое изображение не может модифицироваться пользователем и освобождаться.

cvGetCaptureProperty

```
double cvGetCaptureProperty(  
    CvCapture* capture,  
    int property_id  
);
```

Параметры:

capture

Структура видео-потока.

property_id

Номер свойства, может быть следующим:

CV_CAP_PROP_POS_MSEC

Позиция в миллисекундах фильма с начала файла.



CV_CAP_PROP_POS_FRAMES	Позиция в фреймах (только для видео-файла).
CV_CAP_PROP_POS_AVI_RATIO	Относительное положение в файле (0 – начало, 1 – конец).
CV_CAP_PROP_FRAME_WIDTH	Ширина фреймов в видео-потоке (только для камеры).
CV_CAP_PROP_FRAME_HEIGHT	Высота фреймов в видео-потоке (только для камеры).
CV_CAP_PROP_FPS	Фреймовый показатель (только для камеры).
CV_CAP_PROP_FOURCC	4-х символьный код кодека (только для камеры).

value

Значение свойства.

Функция `cvSetCaptureProperty` устанавливает определенное свойство видео. К настоящему времени функция поддерживает для видео-файлов только: `CV_CAP_PROP_POS_MSEC`, `CV_CAP_PROP_POS_FRAMES`, `CV_CAP_PROP_POS_AVI_RATIO`.

cvCreateVideoWriter

```
CvVideoWriter* cvCreateVideoWriter(
    const char* filename,
    int fourcc,
    double fps,
    CvSize frame_size,
    int is_color=1
);
```

Параметры:

filename

Имя выходного видео-файла.

fourcc

4-х символьный код кодека, используемого для сжатия фреймов. Например, `CV_FOURCC('P','I','M','1')` – MPEG-1 кодек, motion-jpeg кодек – `CV_FOURCC('M','J','P','G')`. Есть два специальных значения, которые могут быть пропущены и ведут себя следующим образом:

`CV_FOURCC_PROMPT` – открывает диалоговое окно, где пользователь может выбрать метод сжатия и его параметры (только Win32);

`CV_FOURCC_DEFAULT` – использует метод сжатия по умолчанию для данного файлового расширения (только Linux).

fps

Частота кадров для созданного видео-потока.

frame_size



Размер фреймов.

is_color

Если не равен нулю, то кодировщик будет кодировать цвета, иначе он будет работать с градациями серого (этот флаг поддерживается только в Windows).

В случае удачного вызова функция cvCreateVideoWriter создаёт структуру видео-записи.

cvWriteFrame

```
int cvWriteFrame(  
    CvVideoWriter* writer,  
    const IplImage* image  
);
```

Параметры:

writer

Структура видеозаписи.

image

Фрейм для записи.

Функция cvWriteFrame записывает/добавляет один фрейм к видео-файлу.

Далее приведены три примера использования видео:

- работа с Web-камерой (листинг 2.2);
- открытие видео-файла и проигрывание его (листинг 2.3);
- сохранение изображения с web-камеры в файл (листинг 2.4).

```
#include "cv.h"  
#include "highgui.h"  
  
int _tmain(int argc, _TCHAR* argv[])  
{  
    CvCapture* capture = 0;  
    IplImage *frame, *frame_copy = 0;  
  
    //Получается видео-поток с камеры  
    capture = cvCreateCameraCapture( 0 );  
  
    //Создаётся окно вывода изображения на экран  
    cvNamedWindow( "VideoOut", 1 );  
  
    if( capture )
```



```

{
    for(;;)
    {
        //Захватывается фрейм
        if( !cvGrabFrame( capture )) break;
        //Получается указатель на изображение
        frame = cvRetrieveFrame( capture );
        if( !frame ) break;
        //Если изображение получается в первый раз, то создаётся
        //копия
        if( !frame_copy )
            frame_copy = cvCreateImage( cvSize(frame->width,
            frame->height),IPL_DEPTH_8U, frame->nChannels );
        //копируется изображение или если нужно отображается его
        //относительно оси X
        if( frame->origin == IPL_ORIGIN_TL )
            cvCopy( frame, frame_copy, 0 );
        else
            cvFlip( frame, frame_copy, 0 );
        //отображается изображение в окно
        cvShowImage( "VideoOut", frame_copy);
        //ожидание нажатия клавиши для завершения
        if( cvWaitKey( 10 ) >= 0 ) goto _cleanup_;
    }
    cvWaitKey(0);

_cleanup_:
    cvReleaseImage( &frame_copy );
    cvReleaseCapture( &capture );
    cvDestroyWindow("VideoOut");
}
return 0;
}

```

Листинг 2.2. Работа с Web-камерой

```

#include "cv.h"
#include "highgui.h"

```



```

//Позиция ползунка
int g_slider_position = 0;
CvCapture* g_capture = NULL;
//Функция обработки выбора позиции
void onTrackbarSlide(int pos) {
    cvSetCaptureProperty(g_capture,CV_CAP_PROP_POS_FRAMES,pos);
}

int _tmain(int argc, _TCHAR* argv[])
{
    cvNamedWindow("Example", CV_WINDOW_AUTOSIZE );
    //Открывается видео-файл
    g_capture = cvCreateFileCapture("test.avi");
    //Определяется общее число фреймов
    int frames = (int) cvGetCaptureProperty(
        g_capture,
        CV_CAP_PROP_FRAME_COUNT
    );
    //Если фреймов больше, чем ноль, то создаётся ползунок
    if( frames!= 0 ) {
        cvCreateTrackbar(
            "Position",
            "Example",
            &g_slider_position,
            frames,
            onTrackbarSlide
        );
    }
    IplImage* frame;
    //Вход в цикл проигрывания фильма
    while(1) {
        //Получение фрейма
        frame = cvQueryFrame( g_capture );
        if( !frame ) break;
        cvShowImage( "Example", frame );
        //Ожидание нажатия ESCAPE

```




```

        char c = cvWaitKey(33);
        if( c == 27 ) break;
    }
    cvReleaseCapture( &g_capture );
    cvDestroyWindow( "Example" );
    return 0;
}

```

Листинг 2.3. Открытие видео-файла и проигрывание его

```

#include "cv.h"
#include "highgui.h"

int _tmain(int argc, _TCHAR* argv[])
{
    CvCapture* capture = 0;
    IplImage *frame, *frame_copy = 0;
    CvVideoWriter* cvVideoWriter=0;

    //Получается видео-поток с камеры
    capture = cvCreateCameraCapture( 0 );

    //Создаётся окно вывода изображения на экран
    cvNamedWindow( "VideoOut", 1 );

    if( capture )
    {
        for(;;)
        {
            //Захватывается фрейм
            if(!cvGrabFrame( capture )) break;

            //Получается указатель на изображение
            frame = cvRetrieveFrame( capture );
            if( !frame ) break;

            //Если изображение получается в первый раз, то создаётся
            //копия
            if( !frame_copy )

```



```

    frame_copy = cvCreateImage( cvSize(frame->width,
        frame->height),IPL_DEPTH_8U, frame->nChannels );

    //Создаётся объект записи
    if (!cvVideoWriter)
        cvVideoWriter=cvCreateVideoWriter("test.avi",
        CV_FOURCC('P','I','M','1'),25,
        cvSize(frame->width,frame->height),1);
    //Копируется изображение или если нужно отображается его
    //относительно оси X
    if( frame->origin == IPL_ORIGIN_TL )
        cvCopy( frame, frame_copy, 0 );
    else
        cvFlip( frame, frame_copy, 0 );
    //Запись фрейма в файл
    cvWriteFrame(cvVideoWriter,frame_copy);
    //Отображается изображение в окно
    cvShowImage( "VideoOut", frame_copy);
    //Ожидание нажатия клавиши для завершения
    if( cvWaitKey( 25 ) >= 0 ) goto _cleanup_;
}
cvWaitKey(0);
_cleanup_:
    cvReleaseVideoWriter(&cvVideoWriter);
    cvReleaseImage( &frame_copy );
    cvReleaseCapture( &capture );
    cvDestroyWindow("VideoOut");
}
return 0;
}

```

Листинг 2.4. Сохранение изображения с web-камеры в файл



3. ОБРАБОТКА И ТРАНСФОРМАЦИЯ ИЗОБРАЖЕНИЙ

3.1. Бинаризация изображений

Бинаризация изображений, т.е. перевод полноцветного или в градациях серого изображения в монохромное, где присутствуют только два типа пикселей (темные и светлые) имеет большое значение при распознавании образов. Особенно это относится к бинарным объектам, таким, как штриховые коды, текст, чертежи и т.п. Существуют различные подходы к бинаризации, которые условно можно разделить на 2 группы:

- пороговые;
- адаптивные.

Если говорить кратко, то пороговые методы бинаризации работают со всем изображением, находя какую-то характеристику (порог), позволяющую разделить все изображение на чёрное и белое. Адаптивные методы работают с участками изображений и используются при неоднородном освещении объектов. Рассмотрим далее, как работают методы бинаризации на примере изображения, представленного на рисунке 3.1.

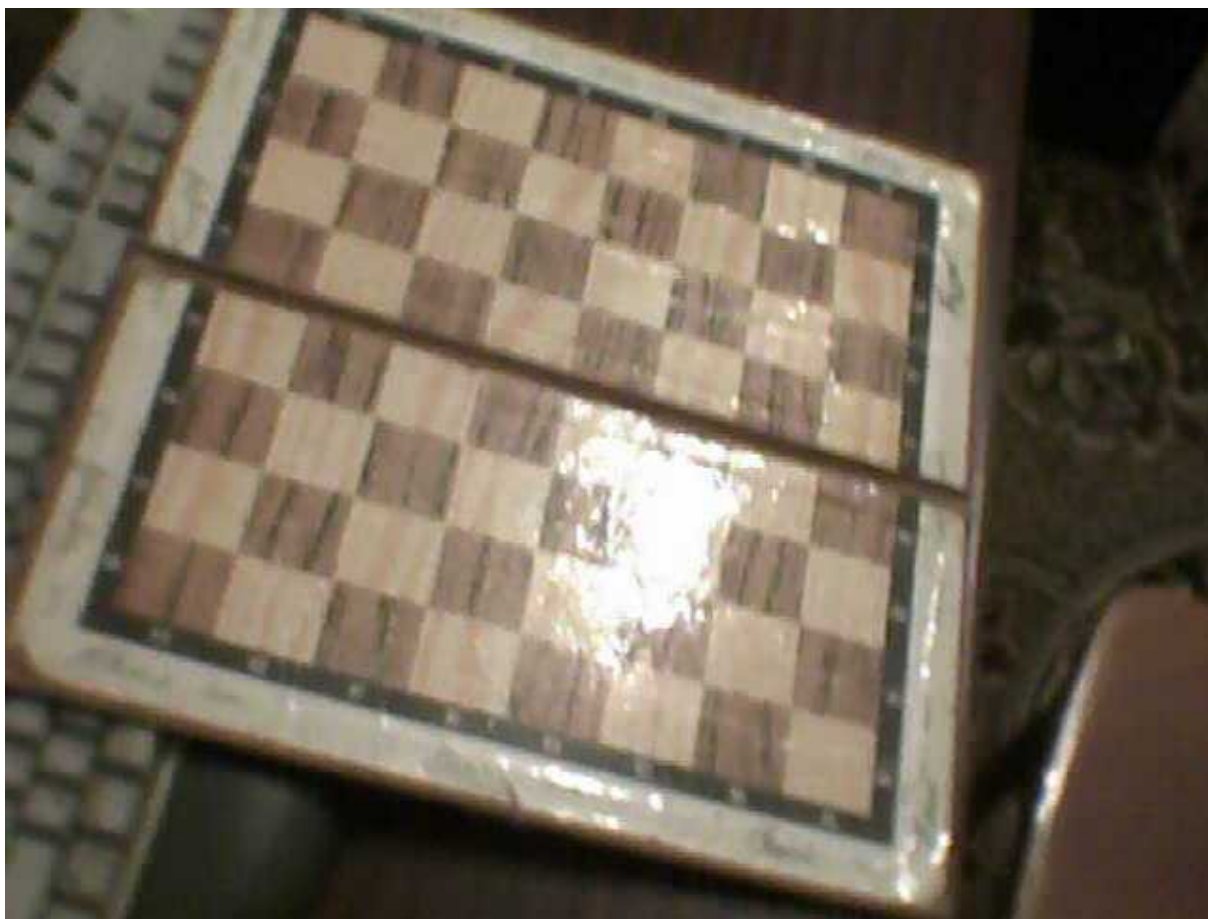


Рис. 3.1. Исходное изображение

Обычно сначала изображение переводится в градации серого (функция `cvtColor`), а затем

выбирается порог яркости. Например, ниже выбран порог 128 яркости:

```
cvThreshold(frame1, frame1, 128, 255, CV_THRESH_BINARY);
```

Результат бинаризации представлен на рисунке 3.2.

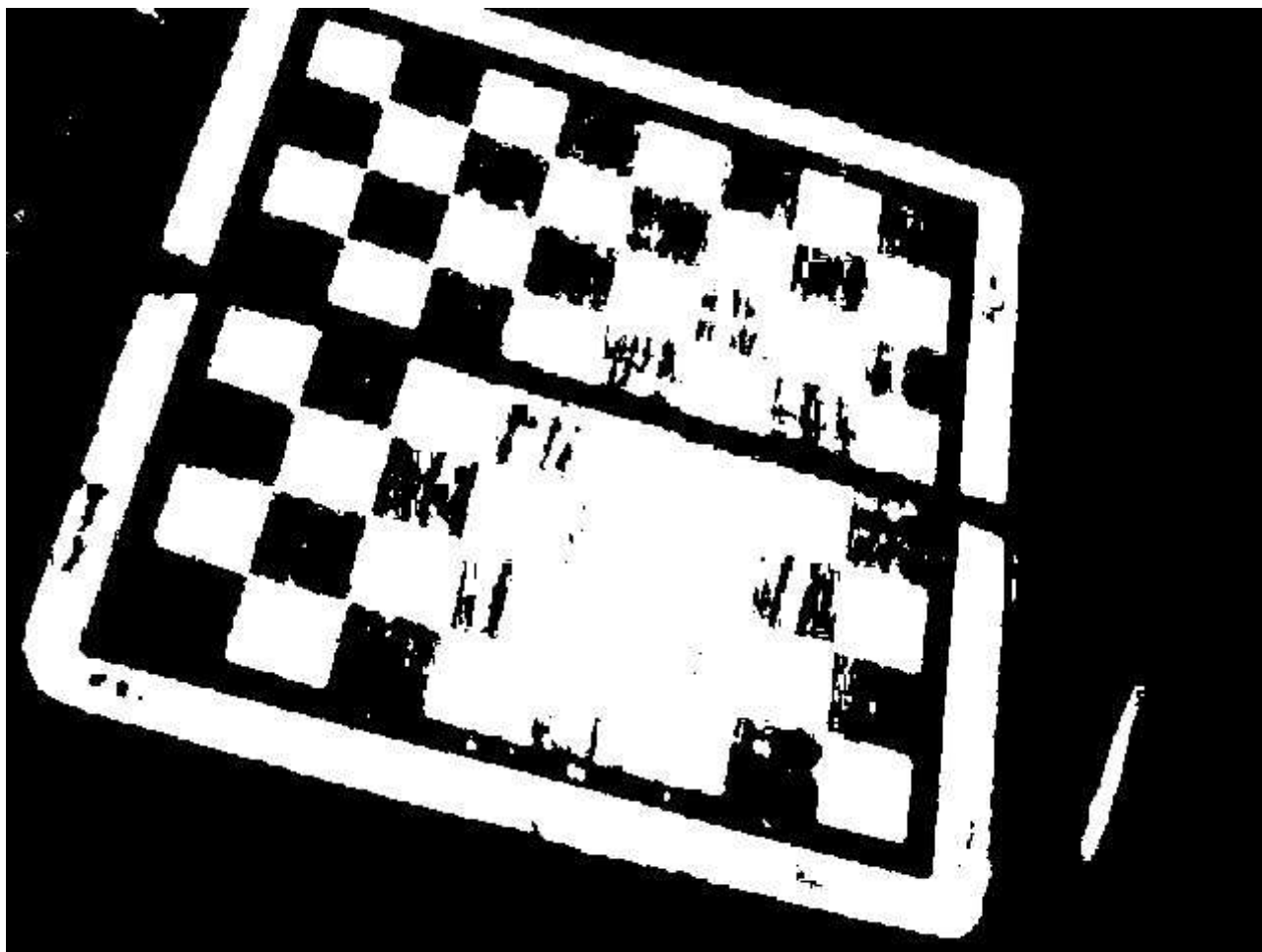


Рис. 3.2. Бинаризация с порогом яркости 128

Понятно, что каждый раз вручную для каждого изображения подбирать свой порог яркости неудобно. Для этого существуют различные критерии бинаризации, например, Отсу, Бернсена, Эйквеля, Ниблэка и т.п. Самым эффективным, как по быстродействию, так и по качеству, считается критерий Отсу. Так, для изображения на рисунке 3.1, критерий Отсу выдал значение оптимального порога яркости равное 104 (Рис. 3.3).

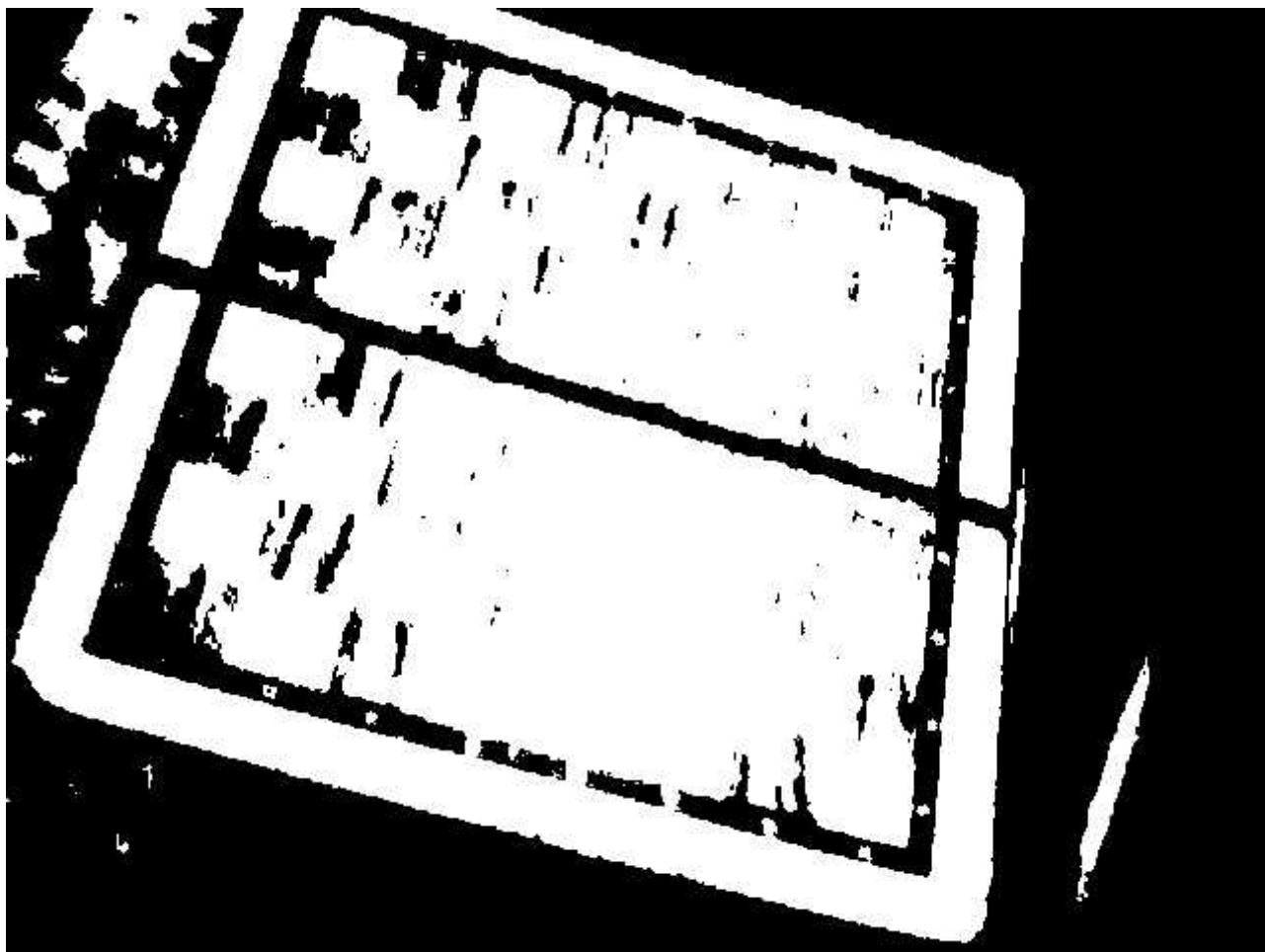


Рис. 3.3. Бинаризация с использованием критерия Отсу

Как видно из рисунка 3, критерий Отсу необходимо использовать с умом. На исходном изображении слишком много темных мест, поэтому порог сдвинулся в темную сторону, и в результате шахматная доска почти полностью стала белого цвета. Если бы анализировалось не всё изображение, а только область доски, то результаты, конечно, были бы другими.

До этого рассматривалась пороговая бинаризация при работе с серым изображением, но ведь можно работать и с полноцветным. Так ниже представлен код функции, который работает с 24-битным изображением, преобразуя его в 8-битное бинаризованное.

```
void Threshold(IplImage* Image, IplImage* gray, CvRect Rect)
{
    uchar* ptr1;
    ptr1 = (uchar*) (Image->imageData );
    uchar* ptr2;
    ptr2 = (uchar*) (gray->imageData );
    int i,j;
    for(i=0; i<Rect.height; i++)
```

```

for(j=0;j<Rect.width;j++)
{
    if (ptr1[(Rect.x+j)*3+0+(Rect.y+i)*Image->widthStep]>100 && ptr1[(Rect.x+j)*3+1+
(Rect.y+i)*Image->widthStep]>140)
        ptr2[j+i*gray->widthStep]=255;
    else
        ptr2[j+i*gray->widthStep]=0;
}
}

```

Результат работы функции представлен на рисунке 3.4. Порог выбран экспериментально (см. условие сравнения ptr1).

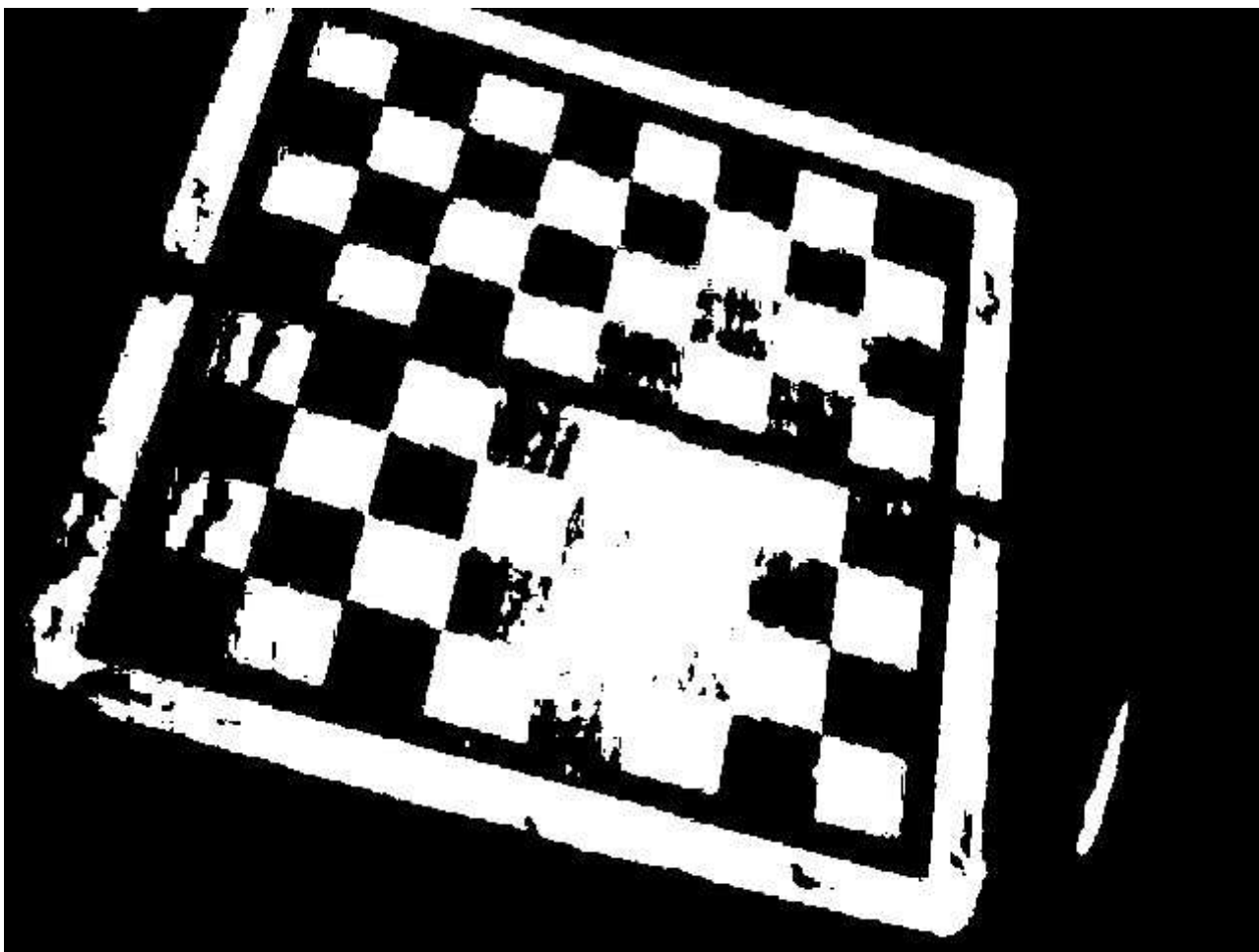


Рис. 3.4. Пороговая бинаризация по 24-битному изображению

Среди адаптивных подходов к бинаризации следует отметить подходы, реализованные в функции cvAdaptiveThreshold. Описание функции приведено ниже.

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

Параметры:

src

Исходное изображение.

dst

Конечное изображение.

max_value

Максимальное значение используемое только с CV_THRESH_BINARY и CV_THRESH_BINARY_INV (ставьте 255).

adaptive_method

Метод CV_ADAPTIVE_THRESH_MEAN_C или CV_ADAPTIVE_THRESH_GAUSSIAN_C.

threshold_type

CV_THRESH_BINARY или CV_THRESH_BINARY_INV.

block_size

Размер окрестностей пикселя, используемых для вычисления порогового значения (3, 5, 7...).

param1

Параметр, зависимый от метода. Для CV_ADAPTIVE_THRESH_MEAN_C и CV_ADAPTIVE_THRESH_GAUSSIAN_C это константа, вычитаемая из среднего значения.

Функция cvAdaptiveThreshold преобразует изображение в градациях серого к монохромному изображению согласно формулам:

CV_THRESH_BINARY

$$dst(x, y) = \begin{cases} \text{max_value} & \text{if } src(x, y) > T(x, y) \\ 0 & \text{otherwise} \end{cases}$$

CV_THRESH_BINARY_INV

$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > T(x, y) \\ \text{max_value} & \text{otherwise} \end{cases}$$

где $T(x, y)$ – порог, рассчитываемый индивидуально для каждого пикселя.

Для метода CV_ADAPTIVE_THRESH_MEAN_C – среднее из $\text{block_size} \times \text{block_size}$ соседних



пикселей минус param1.

Для метода CV_ADAPTIVE_THRESH_GAUSSIAN_C – это весовая сумма (Гауссиан) из $\text{block_size} \times \text{block_size}$ соседних пикселей минус param1.

На рисунках 3.5-3.8 представлены результаты для различных параметров функции.

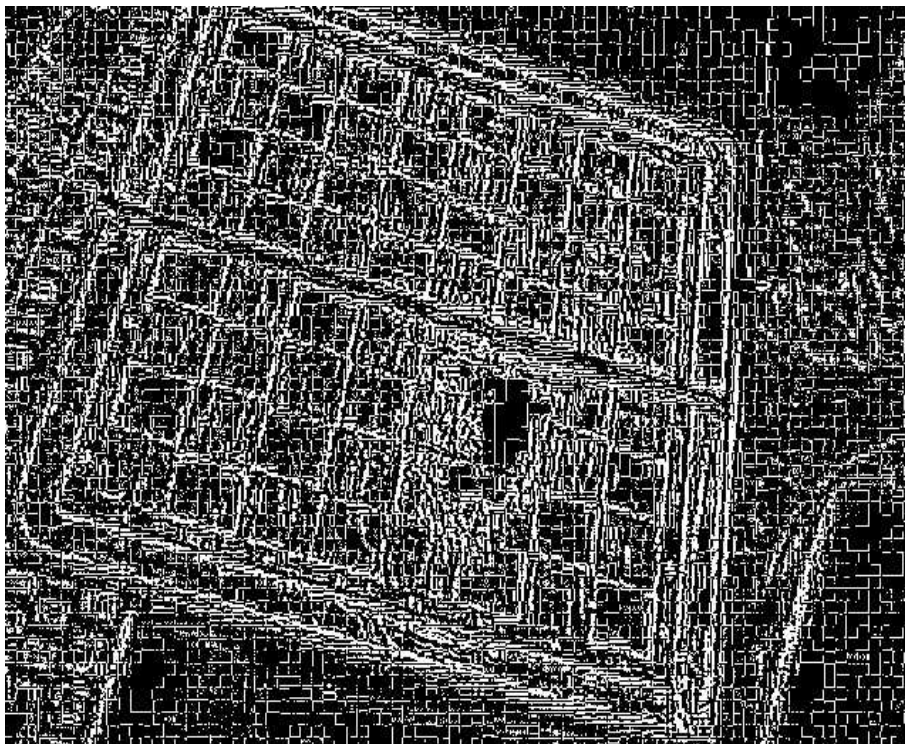


Рис. 3.5. Адаптивное монохромное преобразование (CV_ADAPTIVE_THRESH_MEAN_C, block_size =3, param1=0)

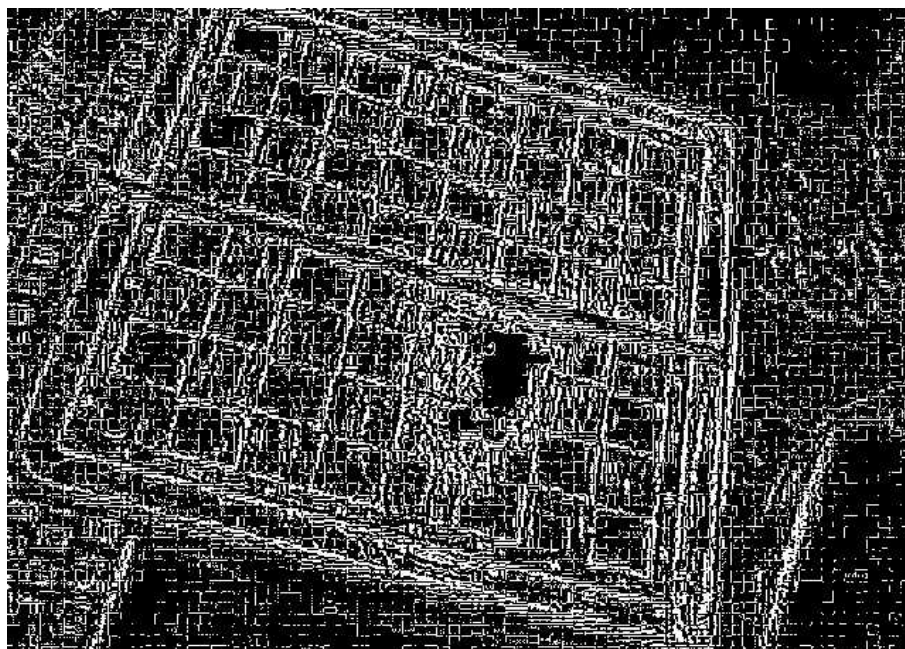


Рис. 3.6. Адаптивное монохромное преобразование (CV_ADAPTIVE_THRESH_GAUSSIAN_C, block_size =3, param1=0)



Рис. 3.7. Адаптивное монохромное преобразование (CV_ADAPTIVE_THRESH_MEAN_C, block_size =25, param1=0)

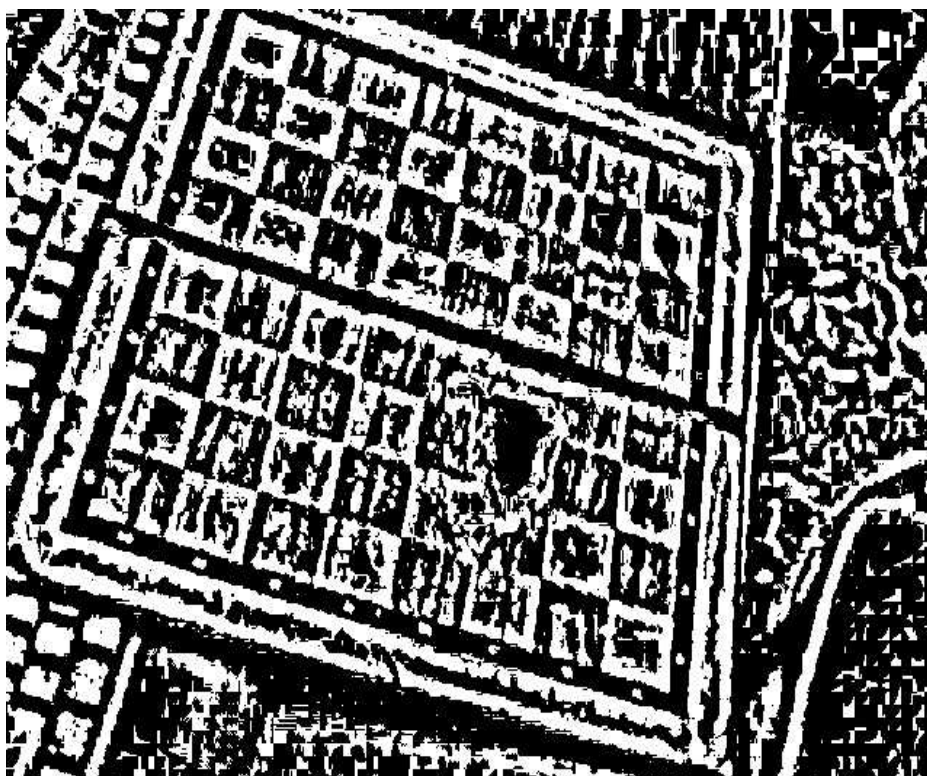


Рис. 8. Адаптивное монохромное преобразование (CV_ADAPTIVE_THRESH_GAUSSIAN_C, block_size =25, param1=0)

Хотелось бы объединить достоинства методов с пороговой и адаптивной бинаризацией. И существует ряд решений для этого. Суть их состоит в том, чтобы с помощью пороговой бинаризации выделить три группы пикселей – белые, чёрные и промежуточные. А уже промежуточные пиксели с помощью адаптивной бинаризации относятся к черному или белому. Результат работы одного такого решения представлен на рисунке 3.9.

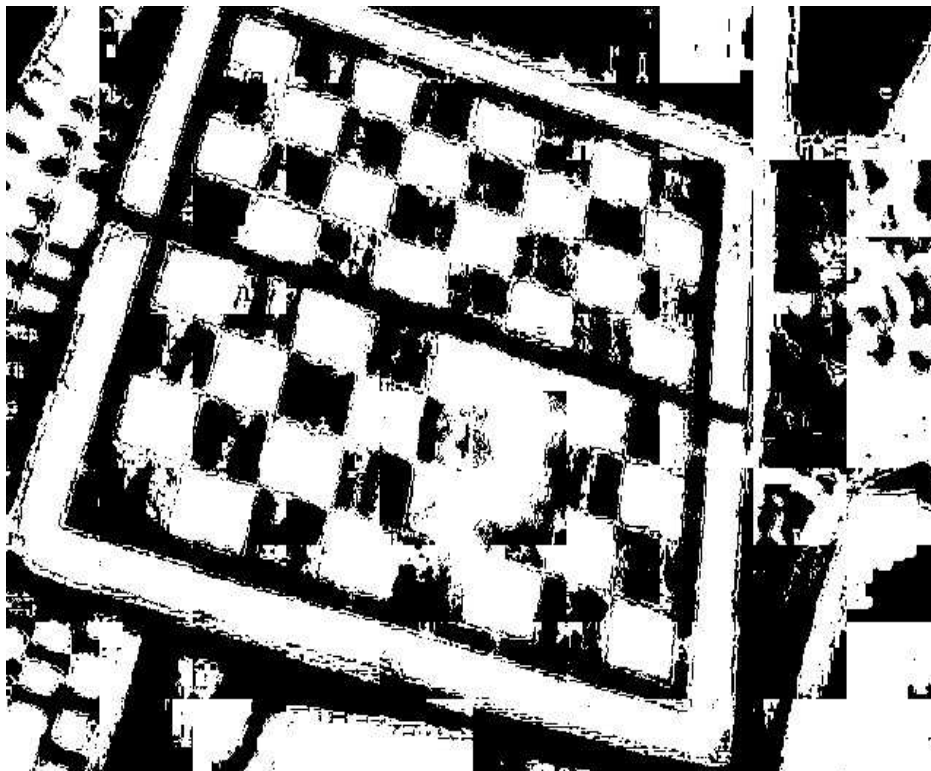


Рис. 3.9. Совмещение подходов адаптивной и пороговой бинаризации

Выбираемый подход бинаризации часто зависит от изображений, которые необходимо распознавать. Например, при распознавании неравномерно освещенного текста можно воспользоваться адаптивными функциями с размерами блоков, превышающими максимальный размер ячейки символа. При этом адаптивные методы бинаризации работают медленнее пороговых, что также влияет на выбор метода.

3.2. Расширенные морфологические преобразования

cvMorphologyEx

```
void cvMorphologyEx(  
    const CvArr* src,  
    CvArr* dst,  
    CvArr* temp,
```

```

    lplConvKernel* element,
    int operation,
    int iterations=1

```

);

Параметры:

src

Входное изображение.

dst

Выходное изображение. Должно быть выделено заранее в том же формате и с теми же размерами, что и **src**.

temp

Дополнительное изображение с теми же параметрами, нужное для некоторых случаев (см. табл. 1).

element

Структурный элемент (задающий окрестность пикселя).

operation

Тип операции (см. табл. 3.1):

Табл. 3.1. Типы морфологических операций

Тип операции	Морфологический оператор	Необходимость temp
CV_MOP_OPEN	Открытие	Нет
CV_MOP_CLOSE	Закрытие	Нет
CV_MOP_GRADIENT	Морфологический градиент	Да
CV_MOP_TOPHAT	Верх шляпы	Только если src=dst
CV_MOP_BLACKHAT	Чёрная шляпа	Только если src=dst

iterations

Количество применённых erode и dilate.

Функция MorphologyEx осуществляет морфологические операции на основе базовых erode и dilate.

erode Эрозия объекта. Приводит к замене значений граничных пикселей объекта на 0. Однократное применение эрозии приводит к удалению слоя границы толщиной в 1 пиксель.

dilate Нарращение объекта. Приводит к замене значений пикселей фона, граничащих с объектом, на 1. Однократное применение наращивания приводит к добавлению к объекту слоя толщиной в 1 пиксель

Открытие



dst=open(src,element)=dilate(erode(src,element),element)

Приводит к соединению областей фона, ранее разъединенных узкими участками пикселей объектов.

Закрытие

dst=close(src,element)=erode(dilate(src,element),element)

Приводит к удалению небольших по площади фрагментов фона внутри объектов, например "дыр" (замкнутых областей фона внутри объекта).

Морфологический градиент

dst=morph_grad(src,element)=dilate(src,element)-erode(src,element)

Верх шляпы

dst=tophat(src,element)=src-open(src,element)

Соответствует вычитанию из исходного изображения результата его открытия.

Чёрная шляпа

dst=blackhat(src,element)=close(src,element)-src

3.2.1. Пример осуществления операции морфологического градиента

В листинге 3.1 приведён текст программы, показывающей как работают морфологические преобразования, а именно морфологический градиент.

```
IplImage* object = cvLoadImage( "1.jpg", 1);
IplImage* object1=cvCreateImage( cvSize(object->width,object->height), 8, 3);
IplImage* temp= cvCreateImage( cvSize(object->width,object->height), 8, 3);
cvNamedWindow("Object", 1);
cvShowImage( "Object", object);
IplConvKernel* structuringElement;
int rows=2;
```



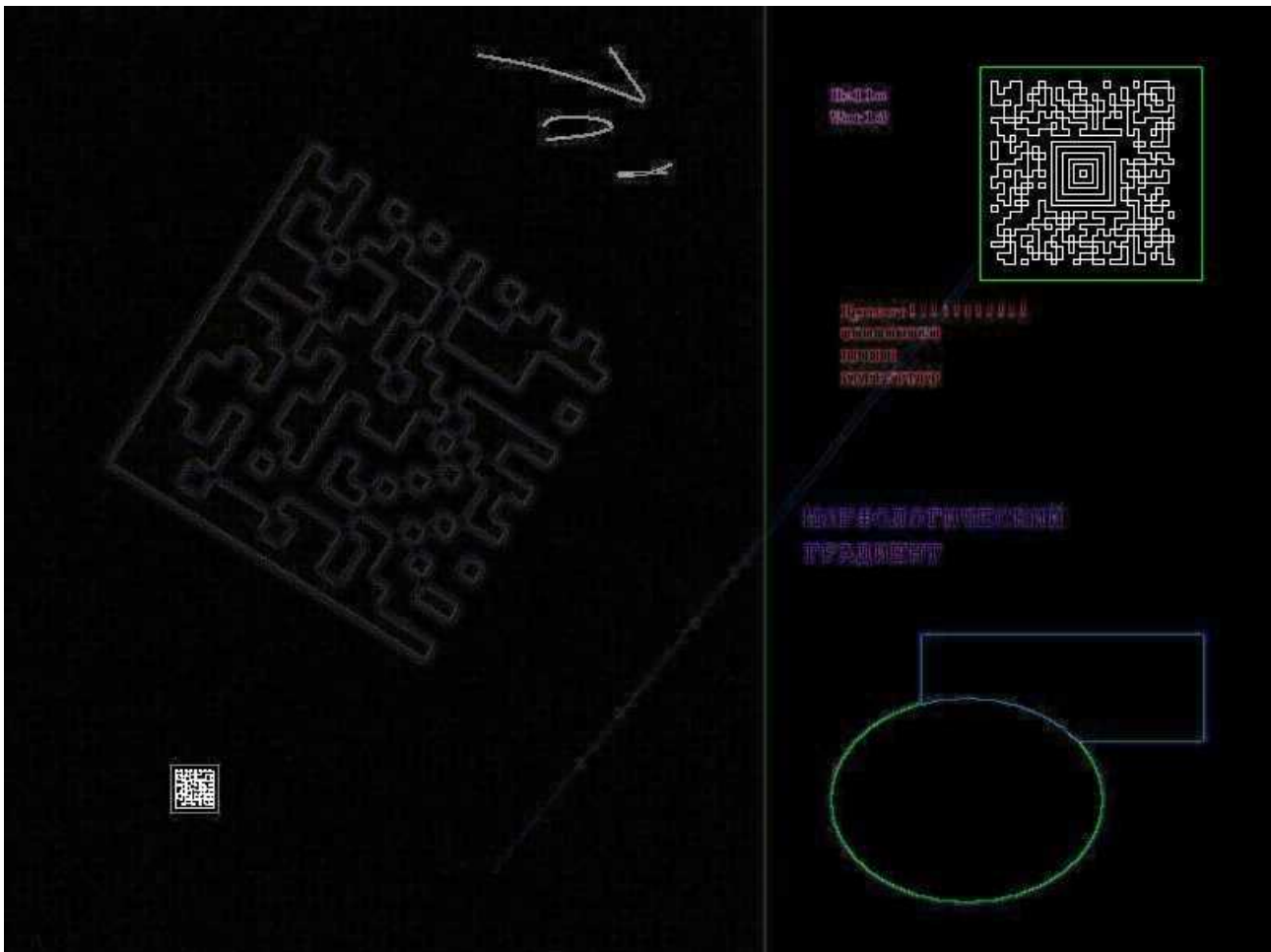


Рис. 3.11. Изображения после трансформации

3.3. Преобразование Фурье

Не буду вдаваться в подробности про то, что такое преобразование Фурье, БПФ, и зачем это нужно. В конце концов, есть Google, Википедия и множество источников, где можно почитать теорию. Напомню только, что данное преобразование позволяет получить спектр сигнала, ну или в обратном случае из спектра – сам сигнал. Преобразование реализуется функцией.

cvDFT

```
void cvDFT(
    const CvArr* src,
    CvArr* dst,
    int flags,
    int nonzeroRows=0
```

);

Параметры:

src

Исходный массив (реальные или комплексные значения).

dst

Выходной массив такой же размерности.

flags

Флаги трансформации.

CV_DXT_FORWARD – прямое преобразование Фурье.

CV_DXT_INVERSE – обратное преобразование.

CV_DXT_SCALE – масштабируемый результат: делится на число элементов массива.

CV_DXT_ROWS – преобразование используется для каждого вектора отдельно. Применяется для обработки многократных векторов одновременно.

CV_DXT_INVERSE_SCALE – CV_DXT_INVERSE вместе с CV_DXT_SCALE.

nonzeroRows

Количество рядов отличных от нуля в исходном множестве. Предназначено для ускорения преобразований. Если значение меньше 1, то игнорируется.

Одномерное преобразование Фурье используется для анализа, например речевых сигналов. В данном случае мы рассмотрим применение двумерного преобразования Фурье для изображений. В качестве примера возьмем исходный код из папки «src» версии OpenCV 2.2. Там немного отличный интерфейс доступа к преобразованию Фурье, но в принципе все тоже самое. Немного переделанный пример представлен ниже.

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

#include <iostream>
using namespace cv;
using namespace std;

void help()
{
    cout << "\nThis program demonstrated the use of the discrete Fourier
transform (dft)\n"
         "The dft of an image is taken and it's power spectrum is displayed.\n"
```



```

        "Call:\n"
        "./dft [image_name -- default lena.jpg]\n" << endl;
}
Mat img;
void DFT();

int main(int argc, char ** argv)
{
    const char* filename = argc >= 2 ? argv[1] : "lena.jpg";

    img= imread(filename, CV_LOAD_IMAGE_GRAYSCALE);
    if( img.empty() )
    {
        help();
        return -1;
    }
    help();
    DFT();
    imwrite("out.jpg",img);

    waitKey();
    return 0;
}

void DFT()
{
    int M = getOptimalDFTSize( img.rows );
    int N = getOptimalDFTSize( img.cols );
    Mat padded;
    copyMakeBorder(img, padded, 0, M - img.rows, 0, N - img.cols,
BORDER_CONSTANT, Scalar::all(0));

    Mat planes[] = {Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F)};
    Mat complexImg;
    merge(planes, 2, complexImg);

```




```

dft(complexImg, complexImg);

// compute log(1 + sqrt(Re(DFT(img))**2 + Im(DFT(img))**2))
split(complexImg, planes);
magnitude(planes[0], planes[1], planes[0]);
Mat mag = planes[0];
mag += Scalar::all(1);
log(mag, mag);

// crop the spectrum, if it has an odd number of rows or columns
mag = mag(Rect(0, 0, mag.cols & -2, mag.rows & -2));

int cx = mag.cols/2;
int cy = mag.rows/2;

// rearrange the quadrants of Fourier image
// so that the origin is at the image center
Mat tmp;
Mat q0(mag, Rect(0, 0, cx, cy));
Mat q1(mag, Rect(cx, 0, cx, cy));
Mat q2(mag, Rect(0, cy, cx, cy));
Mat q3(mag, Rect(cx, cy, cx, cy));

q0.copyTo(tmp);
q3.copyTo(q0);
tmp.copyTo(q3);

q1.copyTo(tmp);
q2.copyTo(q1);
tmp.copyTo(q2);

normalize(mag, mag, 0, 1, CV_MINMAX);

imshow("spectrum magnitude", mag);
normalize(mag, mag, 0, 255, CV_MINMAX);

```



```
mag.copyTo(img);  
}
```

Листинг 3.2

Собственно в примере все действия по преобразованиям перенесены в отдельную функцию DFT. Помимо dft функции (то же, что и cvDft), здесь есть функции для обрезания спектров, перестройки, нормализации и т.д. Что же нам дает такое преобразование? Для этого достаточно посмотреть 3 рисунка.

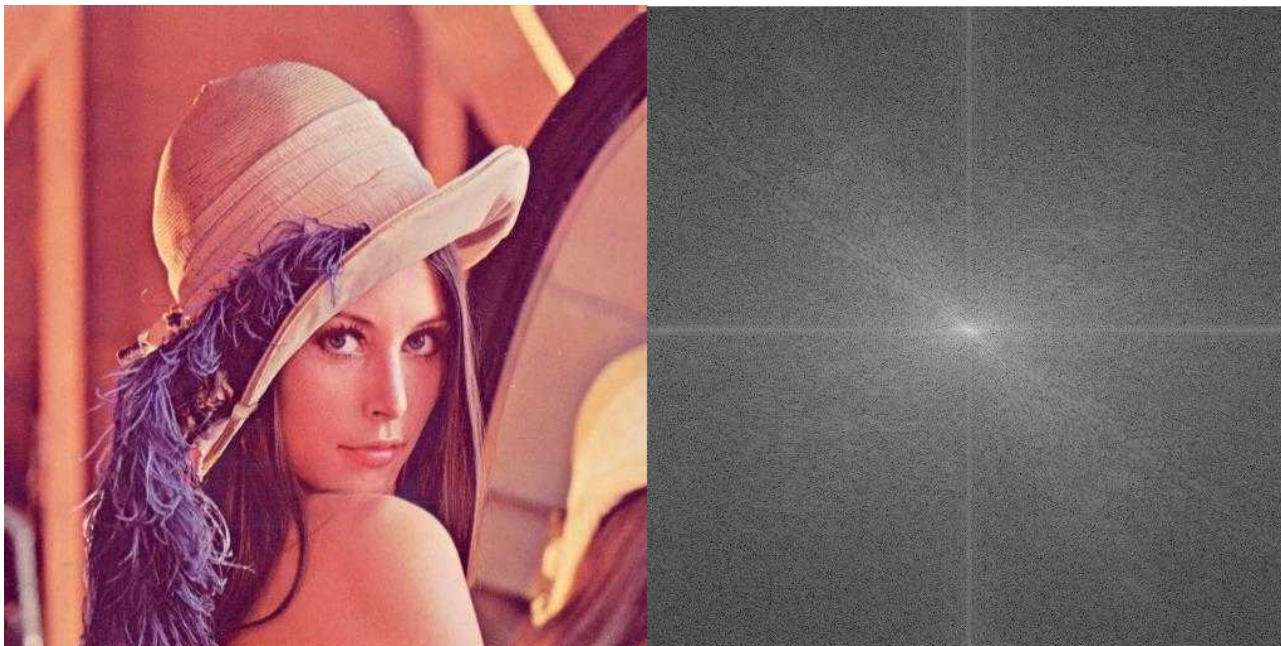


Рис. 3.12

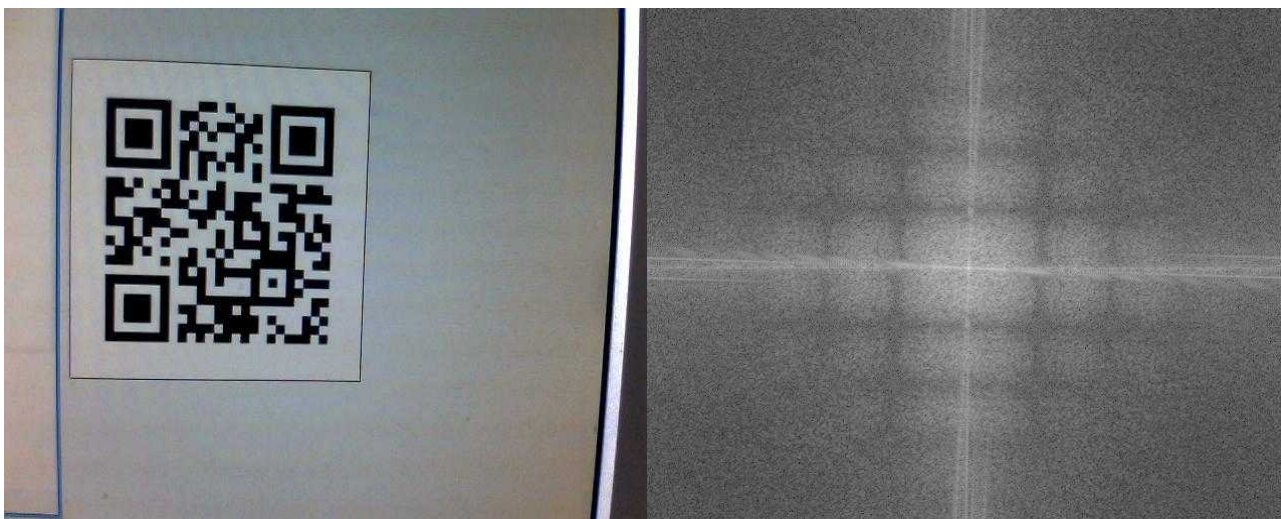


Рис. 3.13



Рис. 3.14

При наличии периодической структуры спектр приобретает явные очертания (про центральный крест мы не говорим, там максимум естественный по природе преобразования Фурье). Причины этого можете почитать в обширных источниках литературы. Нам же важно одно – данное преобразование позволяет выявлять периодические структуры, которое можно использовать для детектирования объектов.

Для интереса видоизмените программу так, как показано ниже.

```
Mat img2= imread(filename, CV_LOAD_IMAGE_GRAYSCALE);
if( img2.empty() )
{
    help();
    return -1;
}
help();
int i,j;
int cc=80;
for(i=0;i<img2.rows/cc;i++)
    for(j=0;j<img2.cols/cc;j++)
    {
        img=Mat(img2, Rect(j*cc, i*cc, cc, cc));
        DFT();
        waitKey();
    }
imwrite("out.jpg",img2);
```

Листинг 3.3

Вы увидите, что участки изображения с периодическими структурами отличаются по спектрам от участков без такой структуры. Так что использование преобразования Фурье для изображений имеет смысл.

3.4. Прозрачный цвет в OpenCV

При разработке различных систем наложения изображений часто бывает необходимо сделать один из цветов прозрачным. Такая возможность поддерживается функцией **cvCopy**. Для этого третьим параметром необходимо передать маску изображения, которую можно получить, переведя исходное изображение в изображение в градациях серого. Пусть у нас есть некоторое изображение в 24 битном формате в `IplImage` – **Image1**. Тогда для копирования изображения туда, куда надо (**Image2**) с прозрачным цветом необходимо выполнить следующие действия:

```
cvSetImageROI(Image2,Rect); //Rect – размеры области
IplImage* image=cvCreateImage(cvGetSize(Image1), 8, 1);
cvCvtColor(Image1,image, CV_BGR2GRAY );
cvCopy(Image1,Image2,image);
cvResetImageROI(Image2);
cvReleaseImage(image);
```

Листинг 3.4

В результате цвет с яркостью 0 – будет прозрачным. Такой метод подходит для прозрачного чёрного цвета.



4. ГИСТОГРАММЫ ЯРКОСТИ

4.1. Гистограммы яркости в OpenCV

Для анализа участков изображений можно использовать гистограммы яркости. В книге «Learning OpenCV» приведён хороший рисунок, иллюстрирующий построение гистограмм, я без изменений приведу его ниже (Рис.4.1).

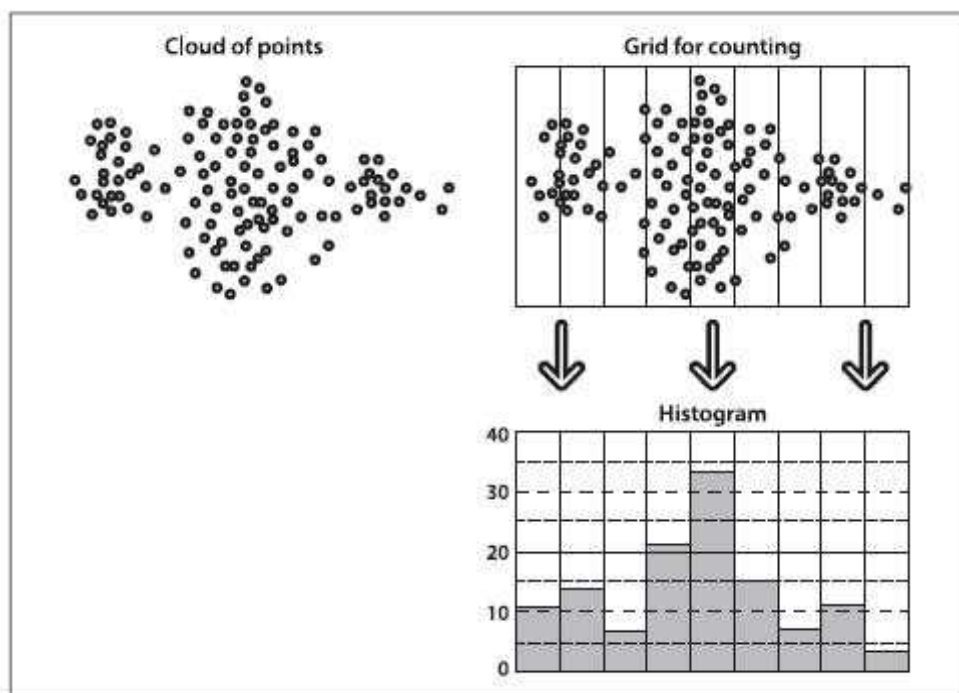


Рис. 4.1. Построение гистограммы

Из рисунка становится совершенно понятно, как строятся гистограммы. Естественно, что гистограммы (в совокупности с другими методами) можно использовать для распознавания изображений. Для получения гистограммы используется функция `cvCreateHist()`.

cvCreateHist

```
CvHistogram* cvCreateHist(  
    int dims,  
    int* sizes,  
    int type,  
    float** ranges=NULL,  
    int uniform=1  
);
```

Параметры:

dims

Число измерений гистограммы.

sizes

Массив, содержащий в себе столбцы гистограммы.

type

Формат гистограммы. CV_HIST_ARRAY или CV_HIST_SPARSE, характеризующие способ хранения гистограммы.

ranges

Массив диапазонов гистограммы, используемый для определения к какому элементу относится входной пиксель.

uniform

Флаг однородности.

Функция создаёт гистограмму указанного размера и возвращает указатель на созданный объект.

Пример создания гистограммы представлен в листинге 4.1.

```
int hist_size = 64;
float range_0[]={0,256};
float* ranges[] = { range_0 };
CvHistogram *hist;
IplImage *hist_image = cvCreateImage(cvSize(320,200), 8, 1);
hist = cvCreateHist(1, &hist_size, CV_HIST_ARRAY, ranges, 1);
cvCalcHist( &gray, hist, 0, NULL );

int bin_w;
bin_w = cvRound((double)hist_image->width/hist_size);
cvSet( hist_image, cvScalarAll(255), 0 );
for( i = 0; i < hist_size; i++ )
    cvRectangle( hist_image, cvPoint(i*bin_w, hist_image->height),
                 cvPoint((i+1)*bin_w, hist_image->height -
                 cvRound(cvGetReal1D(hist->bins,i))),
                 cvScalarAll(0), -1, 8, 0 );

cvNamedWindow( "histogram", 1 );
cvShowImage( "histogram", hist_image );
```

Листинг 4.1. Создание гистограммы



В листинге сначала создаётся картинка, куда будет выводиться гистограмма, затем создаётся объект гистограммы **hist**. После чего высчитывается с помощью функции `cvCalcHist()` сама гистограмма по изображению **gray**, которое должно быть в градациях серого. Если вы хотите построить гистограмму цветного изображения, то вам понадобится 3-х размерная гистограмма по градациям красного, зелёного и синего. После вычисления гистограммы необходимо вывести её в изображение (Рис. 4.2).

Не забываем, что можно использовать для гистограмм не всё изображение, а его участки.

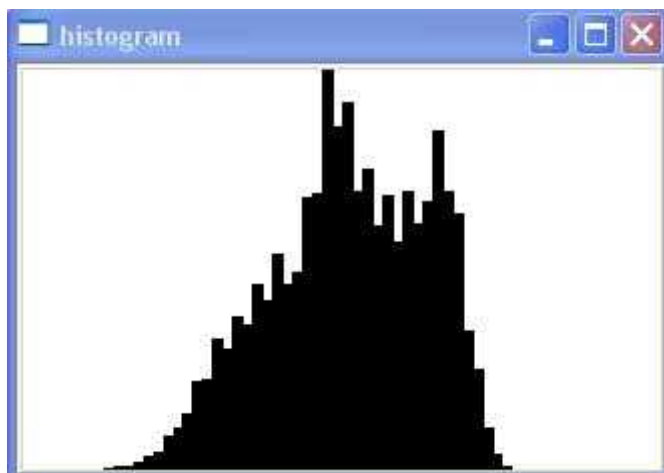


Рис. 4.2. Пример полученной гистограммы

4.2. Сравнение гистограмм

Безусловно, что пользы от гистограмм не будет никакой, если не сравнивать их с эталонными диаграммами. Предположим, есть эталонное изображение – какой-то объект заданных размеров. Также есть множество неизвестных изображений, на которых нужно найти эталонное изображение. Для этого нужно перебирать участки изображений, сравнивая содержимое с эталоном. Можно сравнивать каждую точку из участка, но это будет медленно. Гораздо быстрее по ресурсам – это сравнить гистограммы яркости. Для сравнения гистограмм в OpenCV предусмотрена функция `cvCompareHist()`.

cvCompareHist

```
double cvCompareHist(  
    const CvHistogram* hist1,  
    const CvHistogram* hist2,  
    int method  
);
```

Параметры:

hist1

Первая гистограмма.

hist2

Вторая гистограмма.

method

Метод: CV_COMP_CORREL, CV_COMP_CHISQR, CV_COMP_INTERSECT или CV_COMP_BHATTACHARYYA.

Функция возвращает меру близости одной гистограммы к другой.

Опишем вкратце методы сравнения гистограмм.

CV_COMP_CORREL – корреляционный метод

$$d_{\text{correl}}(H_1, H_2) = \frac{\sum_i H'_1(i) \cdot H'_2(i)}{\sqrt{\sum_i H'^2_1(i) \cdot H'^2_2(i)}}$$

где $H'_k(i) = H_k(i) - (1/N) \left(\sum_j H_k(j) \right)$, N равняется числу столбцов гистограммы.

Возвращаемое значение лежит в интервале $[-1, 1]$, 1 – максимальное соответствие, -1 – максимальное несоответствие, 0 – нет никакой корреляции.

CV_COMP_CHISQR – хи-квадрат

$$d_{\text{chi-square}}(H_1, H_2) = \sum_i \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)}$$

Возвращаемое значение лежит в интервале $[0, \text{неограниченно})$. 0 – максимальное соответствие, а крайнее несоответствие зависит от количества элементов гистограммы.

CV_COMP_INTERSECT – пересечение

$$d_{\text{intersection}}(H_1, H_2) = \sum_i \min(H_1(i), H_2(i))$$

Если гистограммы нормализованы к 1, то совершенное соответствие гистограмм это 1, а совершенное несоответствие – 0.

CV_COMP_BHATTACHARYYA – расстояние Бхатачария

$$d_{\text{Bhattacharyya}}(H_1, H_2) = \sqrt{1 - \sum_i \frac{\sqrt{H_1(i) \cdot H_2(i)}}{\sqrt{\sum_i H_1(i) \cdot \sum_i H_2(i)}}}$$

Совершенное соответствие – это 0, несоответствие – это 1.

На рисунке 4.3 показано сравнение этих методов. Сравниваются гистограммы состоящие из

двух элементов. Эталон – это самая верхняя модель, а 3 нижние – гистограммы полученные с каких-то тестовых образов.

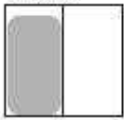
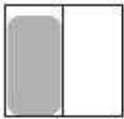

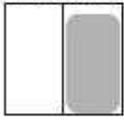
Histograms:		Matching measures:				
Model:		Correlation:	Chi square:	Intersection	Bhattacharyya:	EMD:
						
Exact match:		1.0	0.0	1.0	0.0	0.0
Half match:		0.7	0.67	0.5	0.55	0.5
Total mis-match:		-1.0	2.0	0.0	1.0	1.0

Рис.4.3. Сравнение гистограмм (из книги «Learning OpenCV»)

5. КОНТУРНЫЙ АНАЛИЗ

5.1. Использование контуров

Одним из важнейших методов распознавания графических образов является контурный анализ. Библиотека OpenCV предоставляет возможность разработчикам легко детектировать контуры изображения и манипулировать ими. Для поиска контуров предлагается использовать функцию `cvFindContours()`.

cvFindContours

```
int cvFindContours(  
    CvArr* image,  
    CvMemStorage* storage,  
    CvSeq** first_contour,  
    int header_size=sizeof(CvContour),  
    int mode=CV_RETR_LIST,  
    int method=CV_CHAIN_APPROX_SIMPLE,  
    CvPoint offset=cvPoint(0,0)  
);
```

Параметры:

image

Исходное 8-битное изображение. Отличные от нуля пиксеты обрабатываются как 1, нулевые пиксеты остаются 0 – т.е. изображение является монохромным. Чтобы получить такое изображение, можно использовать `cvThreshold`, `cvAdaptiveThreshold` или `cvCanny`. Функция изменяет исходное содержание изображения.

storage

Контейнер найденных контуров.

first_contour

Указатель на первый найденный контур.

header_size

Размер заголовка последовательности, $> = \text{sizeof}(\text{CvChain})$ если `method=CV_CHAIN_CODE`, и $> = \text{sizeof}(\text{CvContour})$ в противном случае.

mode

`CV_RETR_EXTERNAL` – находятся только критические внешние контуры;

`CV_RETR_LIST` – находятся все контуры, и помещает их в список

`CV_RETR_CCOMP` – находятся все контуры, и записывают их в иерархию с двумя уровнями: верхний уровень – внешние границы компонентов, второй уровень – границы отверстий



CV_RETR_TREE – находятся все контуры, и записывается полная иерархия вложенных контуров.

method

Метод аппроксимации (для всех режимов, кроме CV_RETR_RUNS, который использует встроенную аппроксимацию).

CV_CHAIN_CODE – на выходе очерчивает контур в цепном коде Фримена [1]. Все другие методы выводят многоугольники;

CV_CHAIN_APPROX_NONE – переводит все точки с цепного кода в точки;

CV_CHAIN_APPROX_SIMPLE – сжимает горизонтальные, вертикальные, и диагональные доли;

CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS – применяет одну из разновидностей алгоритма аппроксимации цепочки Teh-Chin.

CV_LINK_RUNS - использует полностью различный алгоритм поиска контура через соединение горизонтальных долей. Только CV_RETR_LIST режим поиска может использоваться с этим методом.

offset

Смещение, с которым каждая точка контура сдвинута. Это полезно, если контуры извлечены из изображения ROI, и затем они должны быть проанализированы в целом контексте изображения.

Функция cvFindContours отыскивает контуры от монохромного изображения и возвращает число найденных контуров. Указатель first_contour заполняется функцией. Он будет содержать указатель на первый наиболее внешний контур или ПУСТОЙ УКАЗАТЕЛЬ, если никакие контуры не обнаружены (если изображение полностью черно). Другие контуры могут быть достигнуты от first_contour, используя h_next и связей v_next. Последовательность контуров описана интересной структурой CvSeq, которая предоставляет интерфейс для всех динамических структур в OpenCV.

CvSeq

```
#define CV_SEQUENCE_FIELDS() \
int flags; /* разные флажки */ \
int header_size; /* размер заголовка последовательности */ \
struct CvSeq* h_prev; /* предыдущая последовательность */ \
struct CvSeq* h_next; /* следующая последовательность */ \
struct CvSeq* v_prev; /* 2-ая предыдущая последовательность */ \
struct CvSeq* v_next; /* 2-ая следующая последовательность */ \
int total; /* общее количество элементов */ \
int elem_size; /* размер элемента последовательности в байтах */ \
char* block_max; /* maximal bound of the last block */ \
char* ptr; /* текущий указатель записи */ \
int delta_elems; /* сколько дополнительных элементов для роста последовательности */ \
CvMemStorage* storage; /* где сохранена последовательность */
```



```
CvSeqBlock* free_blocks; /* свободные блоки в списке*\/  
CvSeqBlock* first; /* указатель на первую последовательность */
```

```
typedef struct CvSeq  
{  
    CV_SEQUENCE_FIELDS()  
} CvSeq;
```

После того, как контуры обнаружены – их можно вывести в изображении с помощью функции `cvDrawContours()`.

cvDrawContours

```
void cvDrawContours(  
    CvArr *img,  
    CvSeq* contour,  
    CvScalar external_color,  
    CvScalar hole_color,  
    int max_level,  
    int thickness=1,  
    int line_type=8,  
    CvPoint offset=cvPoint(0,0)  
);
```

Параметры:

img

Изображение, в которое будут выводиться контуры.

contour

Указатель на первый контур.

external_color

Цвет внешних контуров.

hole_color

Цвет внутренних контуров.

max_level

Максимальный уровень для отображаемых контуров. Если 0, только один контур. Если 1, этот контур и все контуры на том же самом уровне. Если 2, все контуры одинакового уровня и все контуры на один уровень ниже контуров отображаются, и т.д. Если значение отрицательно, функция рисует только дочерние контуры контура до $\text{abs}(\text{max_level})-1$ уровень.

thickness

Толщина контура.



line_type

Тип линии (смотрите описание в функции cvLine).

offset

Сдвигает все координаты точек на указанное значение. Полезно в случае, если, если контуры, найденные в некотором изображении ROI должны быть приняты во внимание для предоставления в изображении.

Функция cvDrawContours рисует контуры в изображении если толщина >=0 или заполняет область, ограниченную контурами если толщина < 0.

Пример использования функции представлен в листинге 5.1.

```
IplImage* image = cvLoadImage( "image.jpg", 1 );

//Создаем изображение в градациях серого
IplImage* img_gray= cvCreateImage( cvSize(image->width,image->height), 8, 1);

CvSeq* contours = 0;
CvMemStorage* storage = cvCreateMemStorage(0);

cvCvtColor( image, img_gray, CV_BGR2GRAY );
cvThreshold( img_gray, img_gray, 128, 255, CV_THRESH_BINARY );
//cvAdaptiveThreshold(img_gray, img_gray, 255, CV_ADAPTIVE_THRESH_GAUSSIAN_C ,
//                    CV_THRESH_BINARY, 21, 7);

cvFindContours( img_gray, storage, &contours, sizeof(CvContour),
                CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );

cvDrawContours( image, contours, CV_RGB(255,0,0), CV_RGB(0,255,0),2, 1, CV_AA,
                cvPoint(0,0) );

cvSaveImage("img2.jpg",image );

cvReleaseImage( &image );
```

Листинг 5.1. Поиск и отображение контуров

Результат выделения контуров при использовании функции cvThreshold() представлен на рисунке 5.1, а при использовании «умного» построения монохромного изображения cvAdaptiveThreshold(), представлен на рисунке 5.2.



Рис. 5.1. Выделение контуров после приведения изображения к монохромному с помощью функции `cvThreshold`

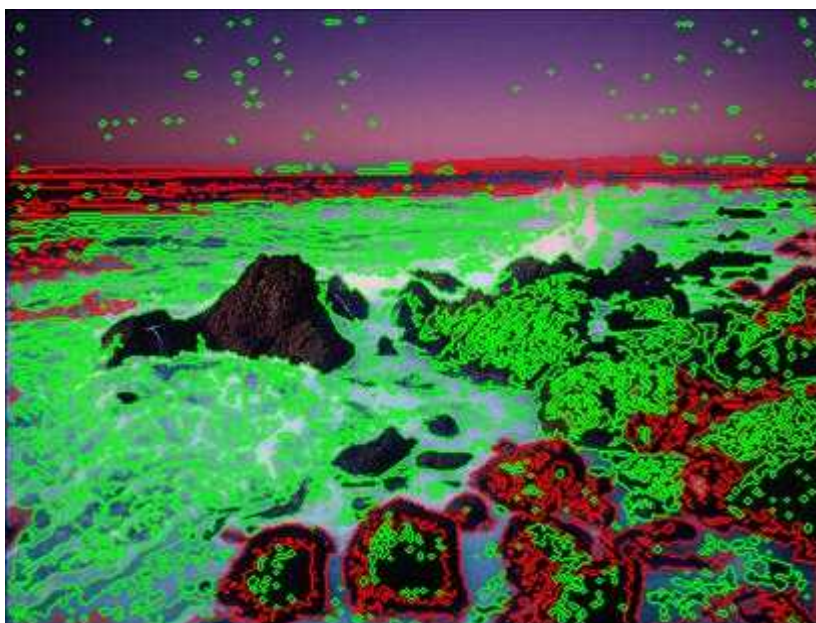


Рис.5.2. Выделение контуров после приведения изображения к монохромному с помощью функции `cvAdaptiveThreshold`

Для сглаживания и получения более аккуратных контуров можно воспользоваться функцией `cvApproxPoly()`.

`cvApproxPoly`

`CvSeq* cvApproxPoly(`

```
const void* src_seq,  
int header_size,  
CvMemStorage* storage,  
int method,  
double parameter,  
int parameter2=0  
);
```

Параметры:

src_seq

Последовательность контуров.

header_size

Размер заголовка аппроксимирующей кривой

storage

Хранилище аппроксимированных контуров.

method

Метод аппроксимации; только CV_POLY_APPROX_DP поддержан, который соответствует алгоритму Дугласа - Пейкера.

parameter

Определенный методом параметр; в случае CV_POLY_APPROX_DP это - желательная точность приближения.

parameter2

Если src_seq - последовательность, это означает, должна ли одиночная последовательность быть аппроксимирована или все последовательности на том же самом уровне или ниже src_seq (см. cvFindContours для описания иерархических структур контура).

Функция cvApproxPoly() аппроксимирует одну или большее количество кривых и возвращает результат приближения. Так если в листинг 5.1. ниже функции cvFindContours() вставить следующую строку:

```
contours = cvApproxPoly( contours, sizeof(CvContour), storage,  
CV_POLY_APPROX_DP, 3, 1 );
```

результат представлен на рисунке 5.3.



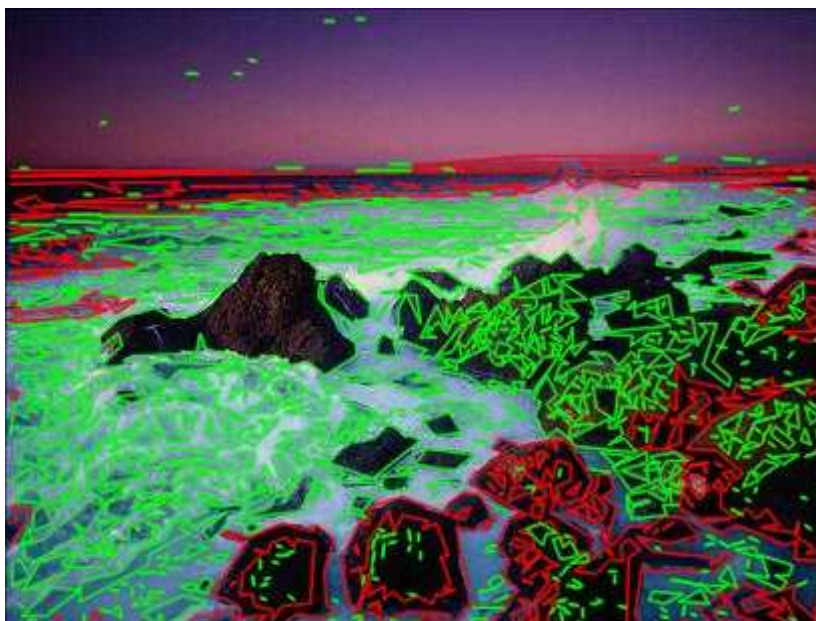


Рис.5.3. Выделение контуров после приведения изображения к монохромному с помощью функции `cvAdaptiveThreshold` и использованию функции `cvApproxPoly`

1. H. Freeman. On the encoding of arbitrary geometric configurations, IRE Transactions on Electronic Computers EC-10(1961) 260-268.

5.1.1. Удаление мелких контуров

При распознавании образов вам могут помешать мелкие контуры. Чтобы удалить их, необходимо пройти по всей цепочке контуров, проверяя размеры каждого и смещая циклы. Для этого в листинге предыдущего параграфа сменить в функции `cvFindContours()` значение `CV_RETR_TREE` на `CV_RETR_LIST` (будет просматриваться последовательный список). И после функции `cvFindContours()` вставить блок кода, представленный в листинге 5.2.

```
CvSeq* h_next=0;
for( CvSeq* c=contours; c!=NULL; c=c->h_next )
{
    if (c!=contours)
    {
        if (c->total<=100) //размер удаляемых контуров
        {
            //удаляем мелкие контуры
        }
    }
}
```

```

        h_next->h_next=h_next->h_next->h_next;
        continue;
    }
}
h_next=c;
}
if (contours->total<=100) contours=contours->h_next;

```

Листинг 5.2 Удаление мелких контуров

Результат работы представлен на рисунке 5.4.

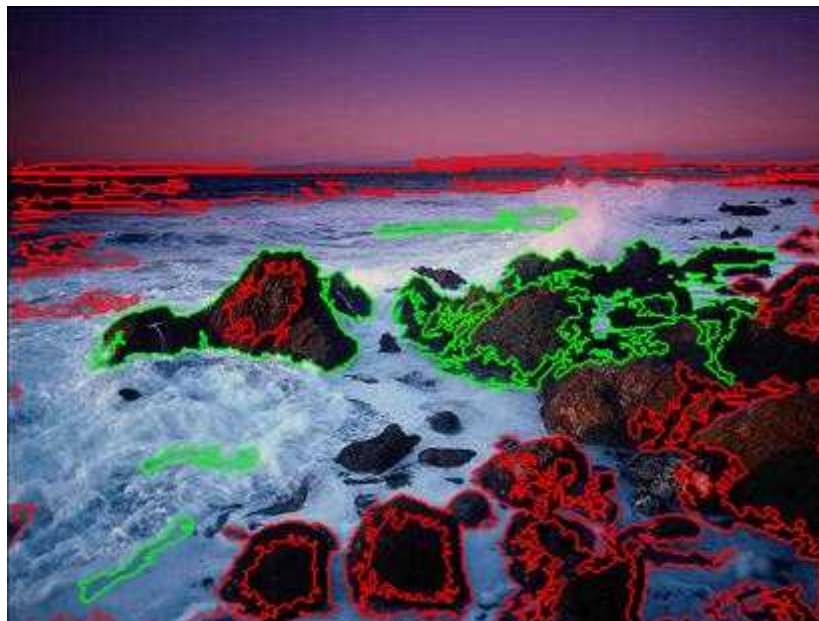


Рис. 5.4. Выделение контуров после приведения изображения к монохромному с помощью функции cvAdaptiveThreshold и удаления мелких контуров

5.2. Моменты в OpenCV

Момент – это характеристика контура, объединённая (суммированная) со всеми пикселями контура. Момент (p,q) определяется как:

$$m_{p,q} = \sum_{i=1}^n I(x,y) x^p y^q$$

Здесь p – порядок x, q – порядок y, где порядок означает, так сказать, мощность, на которой соответствующий компонент взят в сумме с другими отображенными.

Описывает момент структура **CvMoments**.

```
typedef struct CvMoments{  
    // пространственные моменты  
    double m00, m10, m01, m20, m11, m02, m30, m21, m12, m03;  
    // центральные моменты  
    double mu20, mu11, mu02, mu30, mu21, mu12, mu03;  
    // m00 != 0 ? 1/sqrt(m00) : 0  
    double inv_sqrt_m00;  
} CvMoments;
```

В структуре появились ещё и центральные моменты, которые вычисляются по следующей формуле:

$$\mu_{p,q} = \sum_{i=0}^n I(x, y) (x - x_{avg})^p (y - y_{avg})^q$$

где $x_{avg} = m_{10} / m_{00}$ и $y_{avg} = m_{01} / m_{00}$.

Центральные моменты можно нормализовать с помощью функции `cvGetNormalizedCentralMoment()`. Нормализованные моменты вычисляются следующим образом:

$$\eta_{p,q} = \frac{\mu_{p,q}}{m_{00}^{(p+q)/2+1}}$$

Ну и остались ещё инвариантные моменты. Идея здесь в том, что комбинируя различные нормализованные центральные моменты, возможно создавать инвариантные функции, представляющие различные аспекты изображения, которые не зависят от масштабов, вращения и отражения.

В документации к OpenCV приведены следующие формулы:

$$h_1 = \eta_{20} + \eta_{02}$$

$$h_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

$$h_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$h_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$h_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$



$$h_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$

$$h_7 = (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

Но теория теорией, а как применить знания на практике. Помимо изученного в предыдущих параграфах нам понадобится несколько функций.

cvMoments

```
void cvMoments(
    const CvArr* arr,
    CvMoments* moments,
    int binary=0
);
```

Параметры:

arr

Изображение, с которого мы хотим получить моменты (не забываем про то, что это можно применять к участку изображения с помощью ROI).

moments

Сюда будут записываться моменты.

binary

Если этот флаг не равен 0, то все не нулевые пиксели приравниваются к единице.

cvGetHuMoments

```
void cvGetHuMoments(
    CvMoments* moments,
    CvHuMoments* hu_moments
);
```

Параметры

moments

Моменты полученные в предыдущей функции, модифицированные `cvGetNormalizedCentralMoment()`.

hu_moments

Результирующие инвариантные моменты.

. Но как сравнивать моменты? Оказывается для сравнения есть функция `cvMatchShapes`.

cvMatchShapes

```
double cvMatchShapes(
```



```

const void* object1,
const void* object2,
int method,
double parameter = 0
);

```

Параметры:

object1 и **object2** – первый и второй контуры или изображения в градациях серого.

method – метод сравнения.

Ну а последний параметр пока не используется.

Функция сравнивает моменты CvHuMoments. На настоящий момент поддерживаются 3 метода:

CV_CONTOURS_MATCH_I1

$$I_1(A,B) = \sum_{i=1}^7 \left| \frac{1}{m_i^A} - \frac{1}{m_i^B} \right|$$

CV_CONTOURS_MATCH_I2

$$I_2(A,B) = \sum_{i=1}^7 |m_i^A - m_i^B|$$

CV_CONTOURS_MATCH_I3

$$I_3(A,B) = \sum_{i=1}^7 \left| \frac{m_i^A - m_i^B}{m_i^A} \right|$$

Остальные параметры рассчитываются по следующей формуле:

$$m_i^A = \text{sign}(h_i^A) \cdot \log |h_i^A|$$

$$m_i^B = \text{sign}(h_i^B) \cdot \log |h_i^B|$$

Логично предположить, что чем меньше будет возвращаемое значение cvMatchShapes, тем более похожи друг на друга объекты A (1) и B (2).

Однако, вы наверное заметили, что функция cvMatchShapes не принимает параметры CvHuMoments. Тогда зачем, спрашивается, мы считали параметры с помощью cvGetHuMoments. Да и зачем хранить контуры и изображения, если можно хранить моменты. К счастью, есть открытый исходный код – поэтому меняем код как нам нужно, например так.

```
double cvMatchShapesNew( CvHuMoments HuMoments1, CvHuMoments HuMoments2,
    int method)
{
    double ma[7], mb[7];
    int i, sma, smb;
    double eps = 1.e-5;
    double mmm;
    double result = 0;

    ma[0] = HuMoments1.hu1;
    ma[1] = HuMoments1.hu2;
    ma[2] = HuMoments1.hu3;
    ma[3] = HuMoments1.hu4;
    ma[4] = HuMoments1.hu5;
    ma[5] = HuMoments1.hu6;
    ma[6] = HuMoments1.hu7;

    mb[0] = HuMoments2.hu1;
    mb[1] = HuMoments2.hu2;
    mb[2] = HuMoments2.hu3;
    mb[3] = HuMoments2.hu4;
    mb[4] = HuMoments2.hu5;
    mb[5] = HuMoments2.hu6;
    mb[6] = HuMoments2.hu7;

    ...
}
```

Листинг 5.3.

5.2.1. Расчет моментов для символов

Возникает идея использовать моменты для распознавания текста. И в книге «Learning OpenCV» это показано. Ниже приведен листинг, в котором расчеты делаются для всего алфавита. Пусть у нас есть картинка, содержащая алфавит (Рис. 5.5).

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9

Рис. 5.5. Алфавит

```

//1. загрузка картинки
IplImage *image = 0;
image = cvLoadImage( "ex.bmp", 1 );

//2. Поиск контуров
cvNamedWindow( "result2", 1 );
cvNamedWindow( "result1", 1 );
CvMemStorage* storagel = cvCreateMemStorage(0);
CvSeq* contours = 0;
IplImage *image1 = cvCreateImage( cvSize(image->width,image->height), 8,
1 ); ;
cvCvtColor( image, image1, CV_BGR2GRAY );
cvThreshold( image1, image1, 250, 255, CV_THRESH_BINARY );
cvFindContours( image1, storagel, &contours, sizeof(CvContour),
CV_RETR_TREE ,CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );
//делаем CV...TREE т.к. нам нужно получить внешние и внутренние контуры
IplImage* cnt_img = cvCreateImage( cvSize(image->width,image->height), 8,
3 ); ;
CvSeq* _contours = contours;
_contours->h_next=0;
CvSeq* h_next;
CvSeq* contours1=_contours->v_next;
cvShowImage( "result1", image1 );
CvMoments moments;
CvHuMoments HuMoments;
//Создаём файлы, куда будем записывать моменты
FILE *f,*f1,*f2;
f=fopen("data.txt","w");
f1=fopen("data1.txt","w");
f2=fopen("data2.txt","w");
char buf[256];
//Это то, что записано в файле - обучающая выборка
char alfav[]={ "ZYXWVUTSRQPONMLKJIHGFEDCBA9876543210"};
int i=0;
do{
    cvZero( cnt_img );

```



Ниже представлено содержимое файлов.

data.txt

Z: 3.984919e-003 1.287688e-006 2.687221e-014 6.372003e-015 7.911711e-032 -3.302881e-018 -8.338069e-029
Y: 5.148968e-003 2.208064e-006 4.985195e-008 7.690735e-010 4.762011e-018 1.142805e-012 1.624561e-020
X: 4.787206e-003 1.744164e-006 1.204070e-011 1.376395e-011 -1.437680e-023 1.301141e-014 1.766064e-022
W: 5.527161e-003 5.874380e-008 3.635490e-009 6.980813e-010 1.112075e-018 1.691946e-013 -6.105327e-021
V: 4.937904e-003 4.138565e-008 4.693884e-008 1.879257e-012 -5.581365e-022 3.823056e-016 2.666051e-024
U: 4.473054e-003 5.562624e-007 5.405464e-009 2.032533e-010 2.130442e-019 1.515924e-013 -8.213423e-022
T: 4.378969e-003 1.423225e-006 1.902080e-008 7.780806e-010 2.993291e-018 9.282418e-013 -9.912487e-021
S: 3.974653e-003 8.214607e-007 8.779487e-015 3.582633e-015 1.300299e-029 -1.901339e-018 -1.531791e-029
R: 5.115272e-003 8.410446e-007 6.545326e-009 5.466863e-010 4.767339e-019 1.388932e-013 -9.176810e-019
Q: 6.628005e-003 3.173382e-006 1.134230e-009 1.668919e-010 -7.222966e-020 2.105708e-013 -7.433864e-021
P: 5.349237e-003 2.052694e-006 1.284690e-008 4.374450e-010 -5.387050e-019 5.014845e-013 -8.861115e-019
O: 6.248033e-003 1.184363e-007 1.086412e-013 3.006173e-014 -1.367848e-027 8.083947e-018 -1.039446e-027
N: 4.100139e-003 5.614396e-007 4.176275e-010 1.678961e-011 1.360251e-021 -9.846928e-015 3.553609e-022
M: 4.506353e-003 2.376644e-008 1.088658e-011 9.856383e-010 -8.246379e-020 -1.464417e-013 6.019934e-020
L: 5.437270e-003 3.901749e-006 4.263642e-008 1.833126e-009 4.238559e-018 6.032498e-013 -1.564201e-017
K: 4.615566e-003 7.142738e-007 1.442763e-009 3.684745e-010 2.423311e-019 -8.919160e-014 1.159988e-019
J: 5.109741e-003 3.994438e-006 2.305578e-009 1.807177e-010 -1.032055e-019 -1.672179e-013 5.437126e-020
I: 5.640695e-003 8.377988e-006 2.196419e-014 8.129012e-015 2.836185e-029 2.348090e-017 1.048530e-028
H: 4.472321e-003 3.083821e-007 6.908701e-015 4.689146e-015 1.245271e-029 2.459809e-018 2.360623e-029
G: 4.186408e-003 2.886271e-008 2.532166e-010 1.050421e-010 -1.213530e-020 -4.457321e-015 -1.209201e-020
F: 4.379144e-003 2.466410e-006 1.773370e-008 3.403112e-010 5.734369e-019 2.271561e-013 6.083511e-019
E: 3.893560e-003 8.275890e-007 1.456216e-011 5.149285e-011 -1.409969e-021 -4.683854e-014 1.468353e-023
D: 6.204945e-003 1.251260e-006 7.843241e-009 1.582306e-011 -5.107560e-021 -1.697884e-014 2.232638e-021
C: 5.303273e-003 3.228359e-007 3.293276e-009 3.763954e-010 -3.732468e-019 -1.911164e-013 -1.905300e-019
B: 5.978232e-003 8.256085e-007 4.037311e-009 2.266908e-012 2.167894e-022 1.953246e-015 5.867844e-024
A: 5.376205e-003 5.580145e-007 3.783235e-008 8.360696e-010 1.127355e-018 2.942606e-014 -4.564993e-018
9: 5.581510e-003 2.942528e-006 4.606066e-009 8.190901e-010 -3.778130e-019 1.396912e-012 -1.545461e-018
8: 6.416604e-003 1.563235e-006 7.570402e-014 1.246740e-011 1.165450e-023 1.556598e-014 3.298224e-024
7: 5.938822e-003 7.107731e-006 7.459944e-008 6.431850e-009 6.015724e-017 7.093164e-012 -1.273983e-016
6: 5.645375e-003 3.069150e-006 4.892099e-009 8.469374e-010 -1.029830e-019 1.452139e-012 -1.720871e-018
5: 4.484899e-003 2.190228e-006 1.147849e-009 1.497562e-010 4.869946e-020 1.145720e-013 3.851620e-020
4: 5.391954e-003 3.118850e-006 2.744431e-008 2.458470e-009 1.990587e-017 3.168575e-012 -3.399453e-018
3: 5.084270e-003 3.114175e-006 4.489513e-009 4.851954e-010 3.717373e-020 -5.576531e-013 -7.151361e-019
2: 4.723546e-003 1.827829e-006 3.476099e-010 9.531098e-011 2.414414e-022 -1.266137e-013 -1.734676e-020
1: 6.061664e-003 1.323328e-005 8.933567e-009 1.134321e-009 2.538586e-018 3.745031e-012 2.567925e-018
0: 6.624947e-003 1.414681e-006 5.952404e-014 2.525659e-014 -9.752811e-028 2.927483e-017 -8.844502e-029

data1.txt

Z: 2.719600e+004 1.455464e+006 7.978052e+007 4.471696e+009 3.613029e+007 1.933674e+009 1.059970e+011
4.800069e+010 2.569064e+012 6.377244e+013



Y: 2.272100e+004 1.167801e+006 6.173459e+007 3.355010e+009 2.898102e+007 1.489550e+009 7.874351e+010
3.696673e+010 1.899991e+012 4.715410e+013

X: 2.707500e+004 1.451205e+006 8.002229e+007 4.528713e+009 3.315527e+007 1.777118e+009 9.799448e+010
4.060226e+010 2.176292e+012 4.972347e+013

W: 2.741700e+004 1.438320e+006 7.762399e+007 4.304084e+009 3.214692e+007 1.686455e+009 9.101546e+010
3.769482e+010 1.977501e+012 4.420250e+013

V: 2.339400e+004 1.187540e+006 6.168948e+007 3.278798e+009 2.621340e+007 1.330660e+009 6.912414e+010
2.937388e+010 1.491089e+012 3.291686e+013

U: 2.588200e+004 1.343557e+006 7.149321e+007 3.895889e+009 2.765536e+007 1.435613e+009 7.639167e+010
2.955147e+010 1.534040e+012 3.157891e+013

T: 2.415600e+004 1.249811e+006 6.628982e+007 3.603023e+009 2.453086e+007 1.269206e+009 6.731850e+010
2.491246e+010 1.288947e+012 2.530094e+013

S: 2.668600e+004 1.428142e+006 7.815210e+007 4.366836e+009 2.573912e+007 1.377371e+009 7.536865e+010
2.482694e+010 1.328464e+012 2.394816e+013

R: 2.618100e+004 1.427869e+006 7.985620e+007 4.567449e+009 2.388565e+007 1.302899e+009 7.287799e+010
2.179306e+010 1.188952e+012 1.988520e+013

Q: 2.600500e+004 1.484636e+006 8.755681e+007 5.317212e+009 2.239751e+007 1.278911e+009 7.543750e+010
1.929214e+010 1.101790e+012 1.661877e+013

P: 2.212500e+004 1.173237e+006 6.380969e+007 3.552911e+009 1.784583e+007 9.461201e+008 5.144514e+010
1.439531e+010 7.630214e+011 1.161278e+013

O: 2.097000e+004 1.122292e+006 6.151329e+007 3.447266e+009 1.586416e+007 8.490324e+008 4.653579e+010
1.200281e+010 6.423764e+011 9.082290e+012

N: 2.907300e+004 1.549795e+006 8.462428e+007 4.725705e+009 2.047516e+007 1.091315e+009 5.958120e+010
1.442144e+010 7.685457e+011 1.015859e+013

M: 3.199100e+004 1.742450e+006 9.713322e+007 5.529094e+009 2.087054e+007 1.136760e+009 6.336960e+010
1.361807e+010 7.417418e+011 8.887375e+012

L: 2.280500e+004 1.266065e+006 7.197394e+007 4.177439e+009 1.366370e+007 7.590030e+008 4.316931e+010
8.187793e+009 4.550860e+011 4.907117e+012

K: 2.887100e+004 1.545848e+006 8.502336e+007 4.793366e+009 1.583348e+007 8.478996e+008 4.664279e+010
8.685015e+009 4.651609e+011 4.764799e+012

J: 2.279800e+004 1.220576e+006 6.696739e+007 3.757584e+009 1.132153e+007 6.057105e+008 3.321078e+010
5.623332e+009 3.006393e+011 2.793586e+012

I: 1.984800e+004 1.062218e+006 5.852858e+007 3.312253e+009 8.802955e+006 4.711133e+008 2.595850e+010
3.904814e+009 2.089764e+011 1.732337e+012

H: 2.828600e+004 1.513854e+006 8.303209e+007 4.659119e+009 1.110274e+007 5.942132e+008 3.259148e+010
4.359579e+009 2.333225e+011 1.712439e+012

G: 2.939700e+004 1.574788e+006 8.623984e+007 4.820369e+009 1.003646e+007 5.376281e+008 2.944121e+010
3.428298e+009 1.836375e+011 1.171645e+012

F: 2.581100e+004 1.336598e+006 7.096645e+007 3.860836e+009 7.408647e+006 3.832165e+008 2.032304e+010
2.127703e+009 1.099299e+011 6.113951e+011

E: 2.954800e+004 1.581354e+006 8.672796e+007 4.865947e+009 6.970409e+006 3.730427e+008 2.045875e+010
1.645631e+009 8.807080e+010 3.888201e+011

D: 2.124800e+004 1.137316e+006 6.252894e+007 3.523848e+009 3.891206e+006 2.082812e+008 1.144972e+010
7.137557e+008 3.820482e+010 1.311339e+011

C: 2.509900e+004 1.340030e+006 7.338727e+007 4.115049e+009 3.326714e+006 1.775665e+008 9.723995e+009
4.424324e+008 2.360888e+010 5.903862e+010

B: 2.338700e+004 1.258632e+006 6.960383e+007 3.946089e+009 1.872498e+006 1.008613e+008 5.581227e+009
1.513255e+008 8.158754e+009 1.234167e+010

A: 2.439000e+004 1.358637e+006 7.742835e+007 4.501437e+009 6.765950e+005 3.785625e+007 2.165309e+009
2.022144e+007 1.139340e+009 6.436013e+008

9: 2.186300e+004 1.155083e+006 6.276260e+007 3.498015e+009 4.039932e+007 2.134330e+009 1.159658e+011
7.465239e+010 3.943803e+012 1.379491e+014

8: 2.273100e+004 1.191434e+006 6.442915e+007 3.584449e+009 4.081390e+007 2.139240e+009 1.156836e+011
7.328338e+010 3.841110e+012 1.315863e+014



7: 1.830000e+004 9.043640e+005 4.610363e+007 2.423463e+009 3.191267e+007 1.577246e+009 8.041374e+010
5.565186e+010 2.750805e+012 9.705115e+013

6: 2.176300e+004 1.136217e+006 6.106460e+007 3.371652e+009 3.684843e+007 1.923728e+009 1.033857e+011
6.239153e+010 3.257112e+012 1.056425e+014

5: 2.666000e+004 1.395770e+006 7.519284e+007 4.158688e+009 4.371179e+007 2.288465e+009 1.232810e+011
7.167102e+010 3.752160e+012 1.175155e+014

4: 1.943700e+004 1.048955e+006 5.796093e+007 3.270896e+009 3.086312e+007 1.665580e+009 9.203483e+010
4.900681e+010 2.644725e+012 7.781783e+013

3: 2.530300e+004 1.338442e+006 7.297179e+007 4.089183e+009 3.888059e+007 2.056502e+009 1.121110e+011
5.974501e+010 3.159853e+012 9.180751e+013

2: 2.675500e+004 1.422304e+006 7.778456e+007 4.365805e+009 3.970028e+007 2.110487e+009 1.154207e+011
5.891027e+010 3.131713e+012 8.741731e+013

1: 1.904300e+004 1.022936e+006 5.665332e+007 3.224128e+009 2.723548e+007 1.463276e+009 8.105363e+010
3.895294e+010 2.093194e+012 5.571227e+013

0: 2.106100e+004 1.106065e+006 5.982054e+007 3.323638e+009 2.905397e+007 1.525833e+009 8.252333e+010
4.008160e+010 2.104973e+012 5.529650e+013

data2.txt

Z: 1.887617e+006 -5.880542e+003 6.892790e+004 6.710124e+003 1.059719e+006 1.387875e+002 1.153125e+003

Y: 1.712619e+006 5.962101e+006 -3.781172e+001 3.499232e+003 9.455037e+005 -3.804097e+006 -5.863281e+002

X: 2.238509e+006 -4.032637e+005 1.379797e+004 1.840072e+003 1.270779e+006 -8.776046e+003 -1.764231e+005

W: 2.168454e+006 4.342566e+006 -4.032191e+002 9.022726e+002 1.986268e+006 -1.054032e+006 -5.693297e+003

V: 1.406873e+006 4.447790e+006 -2.462873e+002 3.043337e+002 1.295539e+006 -4.562540e+006 -1.238125e+002

U: 1.748007e+006 3.133171e+006 -4.232805e+002 2.740152e+002 1.248393e+006 -1.596740e+006 -2.062039e+003

T: 1.625653e+006 5.024220e+006 -1.672494e+002 2.379868e+003 9.295291e+005 -2.494487e+006 -4.394699e+003

S: 1.722906e+006 -7.506285e+002 -9.748828e+004 3.592572e+001 1.107613e+006 -3.329492e+003 -5.678531e+003

R: 1.982553e+006 -4.027594e+006 2.148203e+005 -4.591957e+005 1.523684e+006 1.527515e+006 1.147852e+006

Q: 2.798342e+006 -9.593725e+005 2.287581e+005 7.197755e+005 1.683913e+006 -2.139029e+005 -1.499672e+006

P: 1.595673e+006 3.305712e+003 -2.023397e+005 -1.721482e+006 1.022862e+006 -1.526049e+006 1.700019e+006

O: 1.449423e+006 -2.895482e+003 -5.797397e+002 8.945010e+002 1.298092e+006 -7.538417e+003 -4.504822e+003

N: 2.009328e+006 4.145778e+005 -1.543033e+005 -4.467775e+005 1.456272e+006 -7.433075e+005 -4.380093e+004

M: 2.227405e+006 -4.089073e+006 7.285404e+003 2.721012e+005 2.384505e+006 -1.563590e+006 7.639744e+005

L: 1.685821e+006 -5.521092e+006 4.357440e+005 -2.511153e+006 1.141930e+006 2.222436e+006 3.163643e+006

K: 2.253588e+006 -3.918272e+005 1.232278e+005 9.543482e+005 1.593646e+006 1.602323e+006 1.479974e+006

J: 1.619297e+006 -1.155158e+006 -4.299366e+005 6.777357e+005 1.036484e+006 2.494861e+005 -1.218782e+006

I: 1.681186e+006 -4.475795e+003 -4.797168e+001 1.884314e+003 5.409272e+005 -5.255662e+002 -2.044465e+003

H: 2.011303e+006 -9.145720e+003 -2.313841e+002 1.163011e+003 1.566991e+006 5.878077e+001 -2.691438e+003

G: 1.878949e+006 -7.812127e+005 -2.199406e+004 3.263426e+005 1.738877e+006 -3.200601e+005 -1.371952e+006

F: 1.751994e+006 4.455539e+006 -4.331888e+005 -1.916350e+006 1.165425e+006 -2.487086e+006 2.070356e+006

E: 2.096833e+006 -8.927690e+003 -6.065494e+002 -4.123880e+005 1.302573e+006 -1.412069e+002 -6.645193e+005

D: 1.653204e+006 -4.161203e+004 1.309219e+003 -1.521810e+006 1.148189e+006 4.929012e+003 1.262611e+006

C: 1.843364e+006 8.550510e+004 -4.605689e+004 1.894082e+006 1.497485e+006 -2.837471e+005 3.200117e+004

B: 1.867284e+006 -8.062717e+005 8.800755e+004 -1.129002e+006 1.402521e+006 6.876506e+005 1.086703e+006

A: 1.745914e+006 -6.194370e+006 1.667501e+005 -1.183345e+006 1.452241e+006 3.659571e+006 2.072657e+006

9: 1.736357e+006 -1.380887e+006 -7.840128e+004 -1.170146e+006 9.315538e+005 -6.400388e+005 1.255822e+006

8: 1.980737e+006 -2.114829e+005 -4.157898e+002 -7.252880e+002 1.334712e+006 -6.348935e+004 -6.394766e+003

7: 1.411044e+006 5.613343e+006 1.603574e+005 -4.545165e+005 5.778077e+005 -2.250782e+006 -9.216060e+005

6: 1.744231e+006 1.423224e+006 -7.877377e+004 1.221513e+006 9.295771e+005 6.101122e+005 -1.205498e+006



5:	2.118052e+006	2.278372e+005	-4.249656e+004	-8.986694e+005	1.069615e+006	9.797706e+005	1.512812e+005
4:	1.352062e+006	-3.009673e+006	-6.834918e+003	2.070806e+006	6.850023e+005	5.908548e+005	-1.086053e+006
3:	2.172790e+006	-6.404669e+005	-1.479175e+005	-1.809737e+006	1.082372e+006	1.299747e+006	-3.345043e+005
2:	2.174447e+006	-4.352553e+005	8.338989e+003	-4.415432e+005	1.206809e+006	5.517269e+005	-6.956066e+005
1:	1.704097e+006	-2.205888e+006	2.627255e+005	-7.114292e+005	4.940800e+005	6.679843e+005	2.188530e+004
0:	1.733089e+006	-3.817790e+003	-1.392664e+002	6.761938e+002	1.205510e+006	-6.347354e+003	-1.828234e+003

Вы можете наблюдать, что моменты различных контуров отличаются. А похожих контуров – схожи (например, ноль и буква «О»). Вы также можете отслеживать поворот контуров (посмотрите 6 и 9). Конечно использование моментов не решает всех проблем, но во многом упрощает распознавание текста.



6. ОТЛИЧИЯ МЕЖДУ КАДРАМИ И СЕГМЕНТАЦИЯ ИЗОБРАЖЕНИЙ

6.1. Выделение отличных от фона объектов

Одной из задач распознавания графических образов является задача выделения на заранее известном неподвижном фоне объектов, которых раньше там не было. В этом параграфе будет рассмотрено, как, к примеру, выделить руку на неподвижном фоне. Программа достаточно простая, она представлена в листинге 6.1.

```
//Необходимо подключить камеру
CvCapture* capture = 0;
capture = cvCaptureFromCAM(0 );

if (!capture) return 1;

cvNamedWindow( "Motion", 1 );

IplImage* image[MAX_CAM_CADR];

CvScalar color;
CvPoint cv1;
CvFont font;
cvInitFont( &font, CV_FONT_HERSHEY_SIMPLEX, 1.3f,
            1.3f,0,1, 8 );

cv1.x=70;   cv1.y=120;
color = CV_RGB(255,0,255);

int i,j;
for(;;)
{
    IplImage* image1;
    if( !cvGrabFrame( capture ) )
        break;
    image1 = cvRetrieveFrame( capture );
```



```

    cvPutText( image1, "Press, when you ready", cv1, &font, color );

cvShowImage( "Motion", image1);

if( cvWaitKey(10) >= 0 ) break;
}

//обучаем фон

for(i=0;i<MAX_CAM_CADR;i++)
{
    if( !cvGrabFrame( capture ))
        break;
    image[i] = cvRetrieveFrame( capture );

    cvShowImage( "Motion", image[i]);

    if( cvWaitKey(20) >= 0 ) break;
}

//Теперь надо посчитать m для каждого

float *m=new float[image[0]->height*image[0]->width*3];

int i1,j1;
uchar* ptr;

for(i=0;i<image[0]->height;i++)
    for(j=0;j<image[0]->width;j++)
        for(j1=0;j1<3;j1++)
        {
            m[(i*image[0]->width+j)*3+j1]=0;
            for(i1=0;i1<MAX_CAM_CADR;i1++)

```




```

        {
            ptr = (uchar*) (image[i1]->imageData);
            m[(i*image[0]->width+j)*3+j1]+=ptr[j*3+i*image[i1]-
>widthStep+j1];
        }
        m[(i*image[0]->width+j)*3+j1]=m[(i*image[0]-
>width+j)*3+j1]/MAX_CAM_CADR;
    }

    //Раз запомнили - можно входить в цикл обработки изображений

    float porog=60.0;
    float k=0;

    for(;;)
    {
        IplImage* image1;
        if( !cvGrabFrame( capture ))
            break;
        image1 = cvRetrieveFrame( capture );

        ptr = (uchar*) (image1->imageData);
        for(i=0;i<image[0]->height;i++)
            for(j=0;j<image[0]->width;j++)
            {
                k=0;
                for(j1=0;j1<3;j1++)
                    k+=abs(m[(i*image1->width+j)*3+j1]-ptr[j*3+i*image1-
>widthStep+j1]);

                if (k<=porog)
                    for(j1=0;j1<3;j1++){
                        if (j1==0) ptr[j*3+i*image1->widthStep+j1]=255;
                        else ptr[j*3+i*image1->widthStep+j1]=0;
                    }
            }
    }

```



```
cvShowImage( "Motion", image1);

if( cvWaitKey(10) >= 0 ) break;
}

//Удаляем все

cvReleaseCapture( &capture );
cvDestroyWindow( "Motion" );

delete m;
```

Листинг 6.1. Выделение новых объектов на известном фоне

Смысл выделения неизвестных объектов заключается в следующем. В буфер **image** запоминается определённое количество кадров первоначального фона. В программе это делается во втором цикле, а в первом ожидается нажатие любой клавиши. Затем, под каждую точку и каждый цвет (R, G и B) выделяется память, в которую будет записываться среднее значение, полученное в нескольких определённых кадрах. После того, как среднее значение подсчитано, программа входит в цикл выделения неизвестных объектов. Здесь **porog** – это пороговое значение, характеризующее отклонение от среднего, выше которого можно считать объект неизвестным. Проверяя каждую точку изображения на отличия от этого порога, выделяем неизвестные объекты. Значения порога зависят от освещения, появления теней и качества камеры. Результат работы программы представлен на рисунке 6.1.

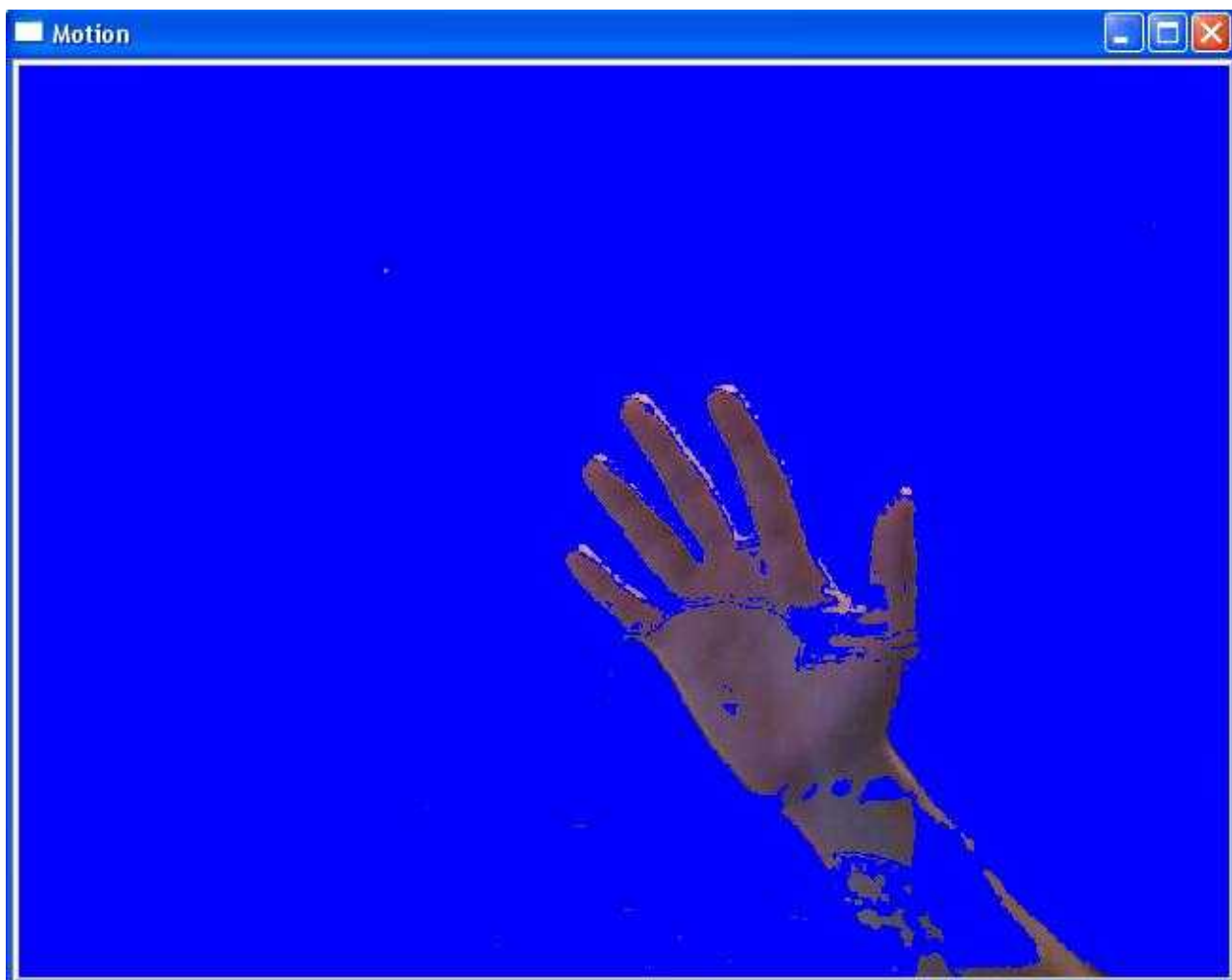


Рис. 6.1. Результат выделения неизвестных объектов на изображении

6.1.1. Движение курсором мышки при помощи пальца или фломастера перед камерой с неоднородным фоном

Мы рассмотрим часть задачи – управление курсором мыши при помощи пальца или другого предмета перед камерой. Из этой задачи, также рассмотрим только частный случай – когда задний фон может быть любым (т.к. если задний фон – это однотонная поверхность, то задача становится очень простой), а также мы не будем рассматривать реальные движения мышью. Т.е. будем автоматически определять кончик пальца и следить за ним.

На рисунке 6.2 представлен фон, на котором будем следить за пальцем.



Рис. 6.2. Фон

Используя принципы выделения отличных от фона объектов, будем находить объект. На рисунке 6.3 представлен объект, который должен детектироваться.



Рис. 6.3. Фон вместе с объектом детектирования

Для того, чтобы детектировать кончик объекта, использовался следующий подход.

1. Всё изображение разделялось на квадраты одинаковых размеров.
2. Внутри каждого квадрата вычислялось, сколько отличных от фона точек.
3. Если больше половины квадрата оказывалось заполненным, то считалось, что в квадрате есть «инородный» объект или его часть.
4. Сверху вниз перебирались все квадраты и, когда находился квадрат, удовлетворяющий условиям, представленным на рисунке 6.4, то считалось, что это и есть кончик объекта.

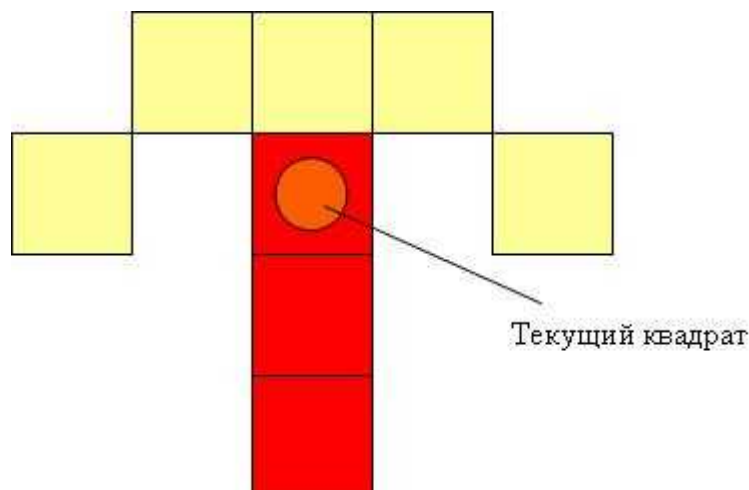


Рис. 6.4. Условия определения кончика объекта (красные квадраты – заполненные, жёлтые – пустые)

Результат работы алгоритма представлен на рисунке 6.5. На кончике выделен толстый красный квадрат.

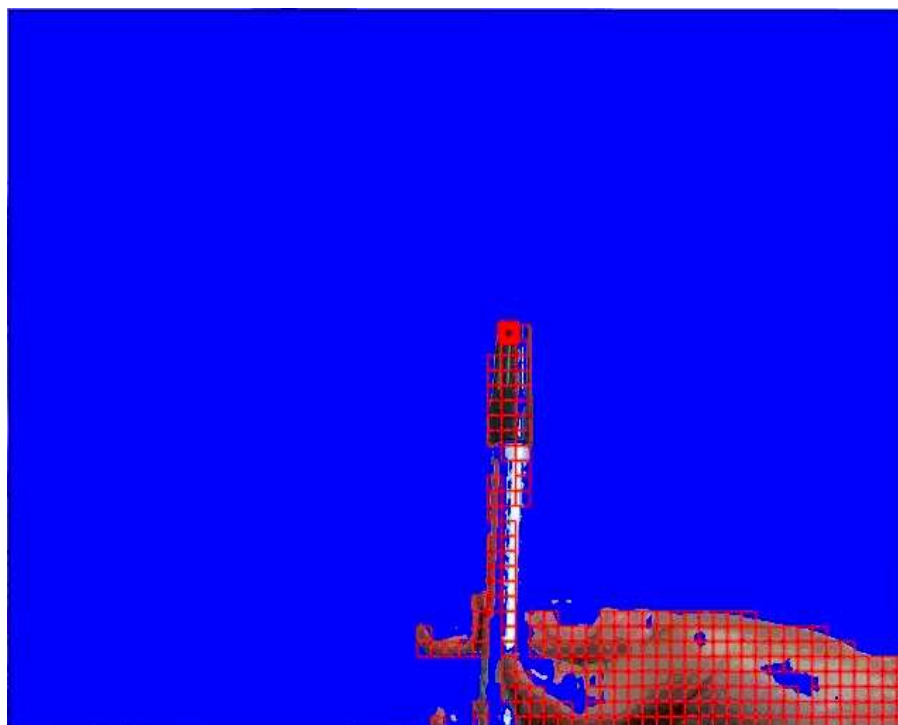


Рис. 6.5. Результат выделения кончика

Текст программы почти полностью сходен с предыдущим параграфом, за исключением того, что алгоритм отсеечения отличных от фона объектов немного был улучшен за счёт включения при обработке фона не одной точки, а соседних. В листинге 6.2 представлена часть программы, отвечающая за выделения квадратов и кончика, которая должна находиться в цикле выделения отличных от фона объектов.

```
ptr = (uchar*) (image1->imageData);

//Считаем матрицу отличных от фона пикселей
for(i=1;i<image[0]->height-1;i++)
    for(j=1;j<image[0]->width-1;j++)
    {
        k=0;
        for(j1=0;j1<3;j1++)
        {
            dd=0;
            for(k1=0;k1<9;k1++)
                dd+=ptr[(j-1+k1%3)*3+(i-1+k1/3)*image[0]->widthStep+j1];
            dd/=9;
            k+=abs(m[(i*image1->width+j)*3+j1]-dd);
        }

        if (k<=porog)
            mat[j+i*image1->width]=1;
        else mat[j+i*image1->width]=0;
    }

//Обнуляем матрицу квадратов
for(i=0;i<image[0]->height/step;i++)
    for(j=0;j<image[0]->width/step;j++)
        mat1[j+width*i]=0;

//считаем заполненность квадратов
for(i=1;i<image[0]->height-1;i++)
    for(j=1;j<image[0]->width-1;j++)
    {
        if (mat[j+i*image1->width])
```

```

    {
        for(j1=0;j1<3;j1++){
            if (j1==0) ptr[j*3+i*image1->widthStep+j1]=255;
            else ptr[j*3+i*image1->widthStep+j1]=0;
        }
    }
else
{
    k11=j/step;
    k12=i/step;
    mat1[k11+k12*width]++;
}
}

//Рисуем квадраты
for(i=0;i<image[0]->height/step;i++)
    for(j=0;j<image[0]->width/step;j++)
        if (mat1[j+width*i]>=step*step/2)
        {
            cv1.x=j*step;cv1.y=i*step;
            cv2.x=(j+1)*step;cv2.y=(i+1)*step;
            cvRectangle(image1,cv1,cv2,color,1,CV_AA,0);
        }

//Определяем зону кончика вверх
k11=0;
for(i=0;i<image[0]->height/step;i++){
    for(j=0;j<image[0]->width/step;j++)
        if (mat1[j+width*i]>=step*step/2 && mat1[j+width*(i+1)]>=step*step/2
        && mat1[j+width*(i+2)]>=step*step/2 &&
            mat1[j+width*(i-1)]<step*step/2 && mat1[j-1+width*(i-1)]<step*step/2 && mat1[j+1+width*(i-1)]<step*step/2 &&
            mat1[j-2+width*(i)]<step*step/2 && mat1[j+2+width*(i-1)]<step*step/2)
        {
            //Нашли кончик
            cv1.x=j*step;cv1.y=i*step;
            cv2.x=(j+1)*step;cv2.y=(i+1)*step;

```




```

        cvRectangle (image1, cv1, cv2, color, 5, CV_AA, 0);
        k11=1;
        x=j*step+step/2;
        y=i*step+step/2;
        break;
    }
    if (k11==1) break;
}
cvShowImage ( "Motion", image1);

```

Листинг 6.2

Далее можно постоянно вызывать функции контроля кончика или использовать технологию трекинга для слежения за точками. К сожалению данный подход хоть и прост, но не обеспечивает достаточное качество слежения для кончика объекта, чтобы его без модернизации можно было прямо сейчас взять и использовать на практике для движения курсором мыши.

6.2. Отличия от фона OpenCV

В OpenCV есть отличный пример, демонстрирующий детектирование отличий от фона `bgfg_segm`. Листинг функции `main` представлен ниже.

```

int main(int argc, char** argv)
{
    IplImage*      tmp_frame = NULL;
    CvCapture*     cap = NULL;
    bool update_bg_model = true;

    if( argc < 2 )
        cap = cvCaptureFromCAM(0);
    else
        cap = cvCaptureFromFile(argv[1]);
    help();

    if( !cap )

```

```

{
    printf("can not open camera or video file\n");
    return -1;
}

tmp_frame = cvQueryFrame(cap);
if(!tmp_frame)
{
    printf("can not read data from the video source\n");
    return -1;
}

cvNamedWindow("BG", 1);
cvNamedWindow("FG", 1);

CvBGStatModel* bg_model = 0;

for( int fr = 1; tmp_frame; tmp_frame = cvQueryFrame(cap), fr++ )
{
    if(!bg_model)
    {
        //create BG model
        bg_model = cvCreateGaussianBGModel( tmp_frame );
        //bg_model = cvCreateFGDStatModel( tmp_frame );
        continue;
    }

    double t = (double)cvGetTickCount();
    cvUpdateBGStatModel( tmp_frame, bg_model, update_bg_model ? -1 : 0 );
    t = (double)cvGetTickCount() - t;
    printf( "%d. %.1f\n", fr, t/(cvGetTickFrequency()*1000.) );
    cvShowImage("BG", bg_model->background);
    cvShowImage("FG", bg_model->foreground);
    char k = cvWaitKey(5);
    if( k == 27 ) break;
    if( k == ' ' )

```



```

{
    update_bg_model = !update_bg_model;
    if(update_bg_model)
        printf("Background update is on\n");
    else
        printf("Background update is off\n");
}
}

cvReleaseBGStatModel( &bg_model );
cvReleaseCapture(&cap);

return 0;
}

```

Листинг 6.3

Как видим, листинг довольно прост. По сути, в цикле опрашивается камера, и на экран выводятся отличия. Попробовав поискать основные элементы, я обнаружил, что в хелпе к `opencv` их нет. Не удивительно, к проекту подключается файл

```
#include <opencv2/video/background_segm.hpp>
```

Важным элементом является структура `CvBGStatModel`.

CvBGStatModel

```

#define CV_BG_STAT_MODEL_FIELDS() \
    int      type; /*тип заднего фона*/ \
    CvReleaseBGStatModel release; \
    CvUpdateBGStatModel update; \
    IplImage* background; /*8UC3 фоновое изображение*/ \
    IplImage* foreground; /*8UC1 изображение переднего плана*/ \
    IplImage** layers; /*8UC3 также отвечает за фон */ \
    int      layer_count; /* количество уровней фона */ \
    CvMemStorage* storage; /*хранилище для объектов переднего плана*/ \
    CvSeq* foreground_regions /*контуры объектов переднего плана*/
typedef struct CvBGStatModel
{
    CV_BG_STAT_MODEL_FIELDS();
} CvBGStatModel;

```

Мы видим, что эта структура полностью определяет отличия от фона. Первоначально структура инициализируется. Причем могут использоваться два способа. `cvCreateFGDStatModel` – инициализация процесса обнаружения на переднем плане, `cvCreateGaussianBGModel` – создает Гауссианов микшер для заднего фона (об этом почитать можно здесь <http://personal.ee.surrey.ac.uk/Personal/R.Bowden/publications/avbs01/avbs01.pdf>) . Как это выглядит, можно посмотреть, перекомпилировав с разными значениями. Ну или пример Гауссианов микшера можно посмотреть в статье:

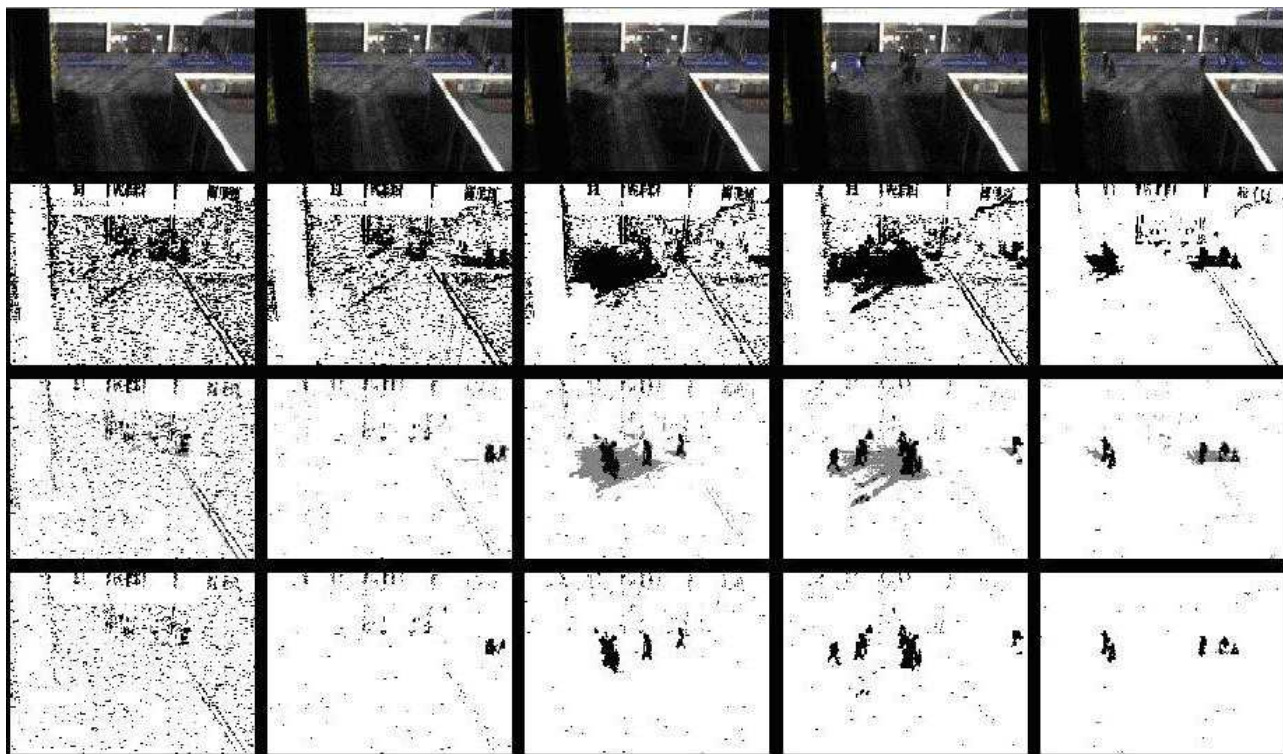


Рис. 6.6

Затем, на каждом кадре производится обновление функцией `cvUpdateBGStatModel`, которая выполняет обновление статической модели и возвращение количества найденных объектов на переднем плане.

6.3. Watershed сегментация

В OpenCV присутствуют функции сегментации изображений, одна из которых реализует watershed (англ. – водораздел) сегментацию.

cvWatershed

```
void cvWatershed(
    const CvArr* image,
```

CvArr* markers

);

Параметры:

image

Входное 8-битное 3-х канальное изображение.

markers

Входное/выходное 32-х битное одноканальное изображение (карта) маркеров.

Функция cvWatershed осуществляет один из вариантов watershed сегментации, описанный в работе [1]. Перед обработкой изображения пользователь должен выделить маркеры (желательные области) на нём.

В листинге 6.4 приведён пример сегментации изображения после поиска контуров, который можно вставить вместо функции DrawContours из листинге про анализ контуров (предварительно удалив маленькие контура).

```
//Инициализация дополнительных параметров
int comp_count=0;
IplImage *image1 = 0;
CvMat* color_tab;
image1=cvCreateImage( cvGetSize(image), IPL_DEPTH_32S, 1 );
cvZero(image1 );

//рисует контура
for( ; contours != 0; contours = contours->h_next, comp_count++ )
{
    cvDrawContours( image1, contours, cvScalarAll(comp_count+1),
        cvScalarAll(comp_count+1), -1, -1, 8, cvPoint(0,0) );
}

//Случайным образом выбираем карту цвета
color_tab = cvCreateMat( 1, comp_count, CV_8UC3 );
CvRNG rng = cvRNG(-1);
int i,j;
for( i = 0; i < comp_count; i++ )
{
    uchar* ptr = color_tab->data.ptr + i*3;
    ptr[0] = (uchar)(cvRandInt(&rng)%180 + 50);
```



```

    ptr[1] = (uchar) (cvRandInt(&rng)%180 + 50);
    ptr[2] = (uchar) (cvRandInt(&rng)%180 + 50);
}

cvWatershed( image, imagel );

// Закрашиваем сегменты изображения
for( i = 0; i < imagel->height; i++ )
    for( j = 0; j < imagel->width; j++ )
    {
        int idx = CV_IMAGE_ELEM( imagel, int, i, j );
        uchar* dst = &CV_IMAGE_ELEM( image, uchar, i, j*3 );
        if( idx == -1 )
            dst[0] = dst[1] = dst[2] = (uchar)255;
        else if( idx <= 0 || idx > comp_count )
            dst[0] = dst[1] = dst[2] = (uchar)0;
        else
        {
            uchar* ptr = color_tab->data.ptr + (idx-1)*3;
            dst[0] = ptr[0]; dst[1] = ptr[1]; dst[2] = ptr[2];
        }
    }
}

```

Листинг 6.4. Пример использования сегментации

В листинге первоначально на чистом изображении рисуются контуры по одиночке, чтобы подсчитать общее количество. Потом случайным образом генерируется карта цвета для закраски различных сегментов изображения. После этого вызывается функция `cvWatershed`. А далее закрашиваются результаты работы функции на основе таблицы цветов. На рисунке 6.7 представлены результаты сегментации изображения после удаления мелких контуров.

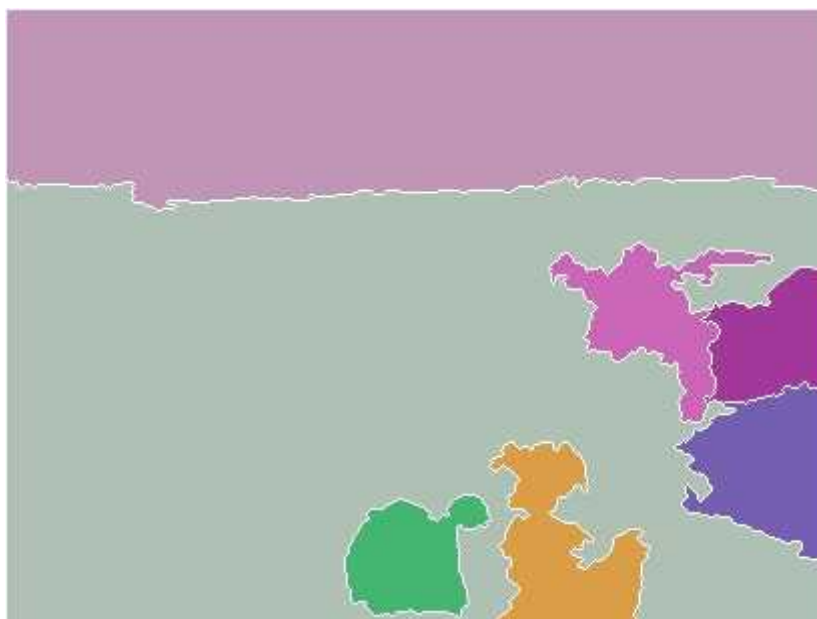


Рис. 6.7. Сегментация watershed

Литература:

1. Meyer, F. (1992). Color image segmentation. In Proceedings of the International Conference on Image Processing and its Applications, pages 303–306.

7. ТРЕКИНГ И ДВИЖЕНИЕ

7.1. Детектирование движений

OpenCV предоставляет возможность детектирования движений на видео-потоке (видео-камера или открытый видео-файл). Рассмотрим пример, идущий в комплекте с OpenCV (Листинг 7.1).

```
#ifndef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
// motion templates sample code
#include "cv.h"
#include "highgui.h"
#include <time.h>
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#endif

// various tracking parameters (in seconds)
const double MHI_DURATION = 1;
const double MAX_TIME_DELTA = 0.5;
const double MIN_TIME_DELTA = 0.05;
// number of cyclic frame buffer used for motion detection
// (should, probably, depend on FPS)
const int N = 4;

// ring image buffer
IplImage **buf = 0;
int last = 0;

// temporary images
IplImage *mhi = 0; // MHI
```



```

IplImage *orient = 0; // orientation
IplImage *mask = 0; // valid orientation mask
IplImage *segmask = 0; // motion segmentation map
CvMemStorage* storage = 0; // temporary storage

// parameters:
// img - input video frame
// dst - resultant motion picture
// args - optional parameters
void update_mhi( IplImage* img, IplImage* dst, int diff_threshold )
{
    double timestamp = (double)clock()/CLOCKS_PER_SEC; // get current time in
seconds
    CvSize size = cvSize(img->width,img->height); // get current frame size
    int i, idx1 = last, idx2;
    IplImage* silh;
    CvSeq* seq;
    CvRect comp_rect;
    double count;
    double angle;
    CvPoint center;
    double magnitude;
    CvScalar color;

    // allocate images at the beginning or
    // reallocate them if the frame size is changed
    if( !mhi || mhi->width != size.width || mhi->height != size.height ) {
        if( buf == 0 ) {
            buf = (IplImage**)malloc(N*sizeof(buf[0]));
            memset( buf, 0, N*sizeof(buf[0]));
        }

        for( i = 0; i < N; i++ ) {
            cvReleaseImage( &buf[i] );
            buf[i] = cvCreateImage( size, IPL_DEPTH_8U, 1 );
            cvZero( buf[i] );

```



```

    }
    cvReleaseImage( &mhi );
    cvReleaseImage( &orient );
    cvReleaseImage( &segmask );
    cvReleaseImage( &mask );

    mhi = cvCreateImage( size, IPL_DEPTH_32F, 1 );
    cvZero( mhi ); // clear MHI at the beginning
    orient = cvCreateImage( size, IPL_DEPTH_32F, 1 );
    segmask = cvCreateImage( size, IPL_DEPTH_32F, 1 );
    mask = cvCreateImage( size, IPL_DEPTH_8U, 1 );
}

cvCvtColor( img, buf[last], CV_BGR2GRAY ); // convert frame to grayscale

idx2 = (last + 1) % N; // index of (last - (N-1))th frame
last = idx2;

silh = buf[idx2];
cvAbsDiff( buf[idx1], buf[idx2], silh ); // get difference between frames

cvThreshold( silh, silh, diff_threshold, 1, CV_THRESH_BINARY ); // and
threshold it
cvUpdateMotionHistory( silh, mhi, timestamp, MHI_DURATION ); // update MHI

// convert MHI to blue 8u image
cvCvtScale( mhi, mask, 255./MHI_DURATION,
            (MHI_DURATION - timestamp)*255./MHI_DURATION );
cvZero( dst );
cvCvtPlaneToPix( mask, 0, 0, 0, dst );

// calculate motion gradient orientation and valid orientation mask
cvCalcMotionGradient( mhi, mask, orient, MAX_TIME_DELTA, MIN_TIME_DELTA, 3 );

if( !storage )
    storage = cvCreateMemStorage(0);

```



```

else
    cvClearMemStorage(storage);

// segment motion: get sequence of motion components
// segmask is marked motion components map. It is not used further
seq = cvSegmentMotion( mhi, segmask, storage, timestamp, MAX_TIME_DELTA );

// iterate through the motion components,
// One more iteration (i == -1) corresponds to the whole image (global
motion)
for( i = -1; i < seq->total; i++ ) {

    if( i < 0 ) { // case of the whole image
        comp_rect = cvRect( 0, 0, size.width, size.height );
        color = CV_RGB(255,255,255);
        magnitude = 100;
    }
    else { // i-th motion component
        comp_rect = ((CvConnectedComp*)cvGetSeqElem( seq, i ))->rect;
        if( comp_rect.width + comp_rect.height < 100 ) // reject very small
components
            continue;
        color = CV_RGB(255,0,0);
        magnitude = 30;
    }

    // select component ROI
    cvSetImageROI( silh, comp_rect );
    cvSetImageROI( mhi, comp_rect );
    cvSetImageROI( orient, comp_rect );
    cvSetImageROI( mask, comp_rect );

    // calculate orientation
    angle = cvCalcGlobalOrientation( orient, mask, mhi, timestamp,
MHI_DURATION);
    angle = 360.0 - angle; // adjust for images with top-left origin

```



```

    count = cvNorm( silh, 0, CV_L1, 0 ); // calculate number of points within
silhouette ROI

    cvResetImageROI( mhi );
    cvResetImageROI( orient );
    cvResetImageROI( mask );
    cvResetImageROI( silh );

    // check for the case of little motion
    if( count < comp_rect.width*comp_rect.height * 0.05 )
        continue;

    // draw a clock with arrow indicating the direction
    center = cvPoint( (comp_rect.x + comp_rect.width/2),
                      (comp_rect.y + comp_rect.height/2) );

    cvCircle( dst, center, cvRound(magnitude*1.2), color, 3, CV_AA, 0 );
    cvLine( dst, center, cvPoint( cvRound( center.x +
magnitude*cos(angle*CV_PI/180)),
                      cvRound( center.y - magnitude*sin(angle*CV_PI/180))), color, 3, CV_AA,
0 );
}
}

int main(int argc, char** argv)
{
    IplImage* motion = 0;
    CvCapture* capture = 0;

    if( argc == 1 || (argc == 2 && strlen(argv[1]) == 1 && isdigit(argv[1][0])) )
        capture = cvCaptureFromCAM( argc == 2 ? argv[1][0] - '0' : 0 );
    else if( argc == 2 )
        capture = cvCaptureFromFile( argv[1] );

    if( capture )
    {

```



```

cvNamedWindow( "Motion", 1 );

for(;;)
{
    IplImage* image;
    if( !cvGrabFrame( capture ))
        break;
    image = cvRetrieveFrame( capture );

    if( image )
    {
        if( !motion )
        {
            motion = cvCreateImage( cvSize(image->width,image->height), 8, 3 );
            cvZero( motion );
            motion->origin = image->origin;
        }
    }

    update_mhi( image, motion, 30 );
    cvShowImage( "Motion", motion );

    if( cvWaitKey(10) >= 0 )
        break;
}

cvReleaseCapture( &capture );
cvDestroyWindow( "Motion" );
}

return 0;
}

#ifdef _EiC
main(1,"motempl.c");
#endif

```

Листинг 7.1



Функцию `update_mhi()` с незначительными модификациями вы можете использовать в вашей программе для детектирования движений. А теперь поподробнее. Видео-поток можно захватывать с камеры `cvCaptureFromCAM()` или из файла `cvCaptureFromFile()`. После чего из потока последовательно выбираются кадр за кадром и передаётся в функцию `update_mhi()`. Параметрами которой являются – исходное изображение, изображение с детектированным движением, параметры для определения монохромности (необходимо для детектирования).

Первоначально в функции проверяется, создан ли ранее указатель на изображение `mhi` и соответствует он изображению поступающего изображения. Если функция вызывается в первый раз, то под изображение выделяется память.

С помощью функции `cvCvtColor()` всё переводится в градации серого. Функция `cvAbsDiff()` вычисляет абсолютное различие между двумя массивами.

AbsDiff

```
void cvAbsDiff(  
    const CvArr* src1,  
    const CvArr* src2,  
    CvArr* dst  
);
```

Параметры:

src1 – первый исходный массив данных,

src2 – второй исходный массив данных,

dst – выходной массив.

Абсолютное различие между двумя массивами вычисляется по формуле:

$$dst(l) = \text{abs}(src1(l) - src2(l)).$$

Все массивы должны иметь тот же самый тип данных и тот же самый размер (или ROI размер).

По выходному массиву с помощью функции `cvThreshold()` строится монохромное изображение, пороговое значение в котором задаются в параметрах функции (см. выше). Функция `cvUpdateMotionHistory()` обновляет изображение истории движения. Далее области движения на картинке преобразуются в выходной синий цвет, а после этого вызывается функция `cvCalcMotionGradient()`, которая вычисляет ориентацию движения. Функция `cvSegmentMotion()` находит все части движения и отмечает их, оценивая каждый (1,2, ...). Всё, детектирование окончено. То, что находится дальше – предназначено для отброса мелких (по площади) движений и вывода на экран областей движений.

Размер и положение области движения вы найдёте в `comp_rect`, а направление движения в `angle`.



7.2. Оптический поток

Вычисляет оптический поток, используя алгоритм Гуннара Фарнебака

```
void calcOpticalFlowFarneback(  
    const Mat& prevImg,  
    const Mat& nextImg,  
    Mat& flow,  
    double pyrScale,  
    int levels,  
    int winsize,  
    int iterations,  
    int polyN,  
    double polySigma,  
    int flags  
);
```

Параметры:

prevImg

1-ое 8-ми битное изображение.

nextImg

2-ое входное изображение такого же размера и типа, что и первое.

flow

Вычисленное изображение потока, имеет тот же размер, что и prevImg и тип CV_32FC2.

pyrScale

Определяет масштаб изображения. pyrScale=0.5 означает классическую пирамиду, где каждый следующий уровень вдвое меньше, чем предыдущий.

levels

Количество уровней пирамиды, включая начальное изображение. Levels=1 означает, что используется только первоначальное изображение.

winsize

Средний размер окна. Большие значения увеличивают ошибкоустойчивость алгоритма, но выдают более размытую область движения.

iterations

Количество итераций алгоритма на каждом уровне пирамиды.

polyN

Размер окрестностей пикселей для нахождения расширенного полиномиала. Большие значения дают сглаженные края.



polySigma

Стандартное отклонение Гауссиана, которое используется для сглаживания в polyN. Для polyN=5 Вы можете устанавливать polySigma=1.1, для polyN=7, хорошее значение было бы polySigma=1.5

flags

Флаги, могут комбинацией из следующих:

OPTFLOW_USE_INITIAL_FLOW – использует входной поток, как начальное приближение потока.

OPTFLOW_FARNEBACK_GAUSSIAN – использует Гауссиан размера winsize* winsize вместо блокового фильтра. Работает медленнее, но точнее.

В листинге 7.2 представлен пример фильтрации видео потока.

```
CvCapture* g_capture = NULL;

void drawOptFlowMap(const Mat& flow, Mat& cflowmap, int step,
                    double scale, const Scalar& color)
{
    double xx=0,yy=0;
    int x,y;
    for(y = 0; y < cflowmap.rows; y += step)
        for(x = 0; x < cflowmap.cols; x += step)
        {
            const Point2f& fxy = flow.at<Point2f>(y, x);
            //Ради теста здесь суммируем все вектора
            xx+=fxy.x;
            yy+=fxy.y;
            line(cflowmap, Point(x,y), Point(cvRound(x+fxy.x), cvRound(y+fxy.y)),
                color);
            circle(cflowmap, Point(x,y), 2, color, -1);
        }
    xx/=x*y;
    yy/=x*y;
}

int _tmain(int argc, _TCHAR* argv[])
```



```

{

cvNamedWindow("Example", CV_WINDOW_AUTOSIZE );

g_capture=cvCreateFileCapture("test.avi");


IplImage* frame;
IplImage* frame1=0;
//Вход в цикл проигрывания фильма
int i=0;

Mat prevgray, gray, flow, cflow;
while(1) {
    //Получение фрейма
    frame = cvQueryFrame( g_capture );
    if( !frame ) break;
    cvShowImage( "Example", frame );
    if (frame1==0)
    {
        //Уменьшаем размеры изображения для убыстрения фильтра
        CvSize c=cvGetSize(frame);
        c.height/=2;
        c.width/=2;
        frame1=cvCreateImage(c, 8, 3);

    }
    cvResize(frame,frame1,0);
    cvtColor(frame1, gray, CV_BGR2GRAY);
    if( prevgray.data )
    {
        calcOpticalFlowFarneback(prevgray, gray, flow, 0.5, 3, 15, 3, 5, 1.2,
0);
        cvtColor(prevgray, cflow, CV_GRAY2BGR);
        drawOptFlowMap(flow, cflow, 16, 1.5, CV_RGB(0, 255, 0));
    }
}
}

```



```

    imshow("flow", cflow);

}
//Ожидание нажатия ESCAPE
char c = cvWaitKey(10);
if( c == 27 ) break;
i++;
//onTrackbarSlide(i);
std::swap(prevgray, gray);
}
cvReleaseCapture( &g_capture );
cvDestroyWindow( "Example" );

return 0;
}

```

Листинг 7.2

7.3. Управляем презентацией взмахом руки

Наверняка многим из вас приходило в голову управлять действиями компьютера с помощью, например, движения рук. Хорошо было бы управлять переключением слайдов презентации на расстоянии без использования специализированных средств, а только при наличии обычной Web-камеры. В листинге 7.3 представлен код функции, позволяющей определить взмах руки и эмулировать нажатие клавиши.

```

void FoundMotion(IplImage* img1, IplImage* img2)
{
    //Установка начальных параметров
    short VKCode = ' ';
    CvSize img_sz = cvGetSize( img1 );
    int win_size = 10;

    IplImage* eig_image = cvCreateImage( img_sz, IPL_DEPTH_32F, 1 );
}

```



```

IplImage* tmp_image = cvCreateImage( img_sz, IPL_DEPTH_32F, 1 );

int corner_count = MAX_CORNERS;

CvPoint2D32f* cornersA = new CvPoint2D32f[ MAX_CORNERS ];

cvGoodFeaturesToTrack(img1,eig_image,tmp_image,ornersA,&corner_count,
                      0.01,5.0,0,3,0,0.04);

cvFindCornerSubPix(img1,ornersA,corner_count,cvSize(win_size,win_size),
                  cvSize(-1,-1),cvTermCriteria(CV_TERMCRIT_ITER|
CV_TERMCRIT_EPS,20,0.03));

//Создаются дополнительные массивы для cvCalcOpticalFlowPyrLK
char features_found[ MAX_CORNERS ];
float feature_errors[ MAX_CORNERS ];
CvSize pyr_sz = cvSize( img1->width+8, img2->height/3 );
IplImage* pyrA = cvCreateImage( pyr_sz, IPL_DEPTH_32F, 1 );
IplImage* pyrB = cvCreateImage( pyr_sz, IPL_DEPTH_32F, 1 );
CvPoint2D32f* cornersB = new CvPoint2D32f[ MAX_CORNERS ];

cvCalcOpticalFlowPyrLK(img1,img2,pyrA,pyrB,ornersA,ornersB,corner_count,cvSize
( win_size,win_size ),5,features_found,feature_errors,cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, .3 ),0);

int all=0;
int up=0;
int down=0;
F_LINE f;
//Совпадения с точками нашли, можно их обрабатывать
for( int i=0; i<corner_count; i++ ) {
    if( features_found[i]==0 || feature_errors[i]>550 ) {
        continue;
    }
    CvPoint p0 =
cvPoint(cvRound( ornersA[i].x ),cvRound( ornersA[i].y ));

```



```

        CvPoint p1 =
cvPoint( cvRound( cornersB[i].x ), cvRound( cornersB[i].y ) );

        //Удаляем точки с небольшим смещением
        if (sqrt( (double)pow( (double)p0.x-p1.x,2) + (double)pow( (double)p0.y-
p1.y,2) ) < 10) continue;

        //Делаем формулу линии
        all++;
        f = MakeLine( p0.x, p0.y, p1.x, p1.y );
        //Только вертикальные линии
        if (f.b == 2 && absf(f.b1) < 0.2)
        {
            if (p1.y < p0.y) down++;
            else up++;
            cvLine( img2, p0, p1, CV_RGB(255,0,0), 2 );
        }
    }
    cvShowImage( "M", img2 );

    cvReleaseImage( &eig_image );
    cvReleaseImage( &tmp_image );
    cvReleaseImage( &pyrA );
    cvReleaseImage( &pyrB );
    delete cornersA;
    delete cornersB;

    if (motion_1 > 0) motion_1--;
    if (motion_1 == 10) motion_1 = 0;

    if (all > 20)
    if ((double)up/all > 0.4 && (double)down/all < 0.1)
    {
        //объект движется вверх
        if (motion_1 < 10) motion_1 = 10;
    }

    if (all > 20)

```

```

    if ((double)down/all>0.4 && (double)up/all<0.1)
    {
        //объект идёт вниз
        if (motion_1>0 && motion_1<8) {
            printf("vzmax22222222\n");
            //Состоялся взмах
            motion_1=15;
            VKCode=VK_RIGHT;
            keybd_event(VKCode, 0, 0, 0);
        }

    }

    printf("%d;%d;%d\n", all, up, down);
}

```

Листинг 7.3. Функция детектирования взмаха руки

Листинг описан поверхностно – без подробного описания назначения функций. Часть кода взята из известной книги «Learning OpenCV» [1].

Функция получает на входе два изображения – текущий и предыдущий кадр с камеры. Ранее должно быть определено:

```
const int MAX_CORNERS = 500;
```

Первоначально выделяется память под хранение углов **cornersA**. Затем с помощью функции `cvGoodFeaturesToTrack()` находятся углы с наибольшим собственным значением в изображении. С помощью функции `cvFindCornerSubPix()` определяется точное местоположение углов или радиальных седловых точек.

После создания дополнительных массивов вызывается функция `cvCalcOpticalFlowPyrLK()`, которая осуществляет алгоритм слежения за точками Lucas-Kanade tracker. Подробнее об этом алгоритме вы можете почитать в публикации [2] или в описании OpenCV.

В результате работы функции находятся совпадения с точками на втором изображении. Затем идёт обработка полученных результатов. В функции используются дополнительные переменные, структура и функция (Листинг 7.4).

```

//Переменная для отслеживания комбинации движения
int motion_1=0;

struct F_LINE

```

```

{
    int k1;
    int b;
    double b1,b2;
};

F_LINE MakeLine(double x1,double y1,double x2,double y2)
{
    F_LINE f;
    f.k1=3;
    //вычисляем координаты линии от i3 к i4
    f.b=1;
    if (absf(x1-x2)<absf(y1-y2)) f.b=2;
    if (f.b==1)
    {
        //y=f(x);
        f.b1=(double) (y2-y1) / (x2-x1);
        f.b2=(double) y1- (double) x1*f.b1;
    }
    if (f.b==2)
    {
        //x=f(y);
        f.b1=(double) (x2-x1) / (y2-y1);
        f.b2=(double) x1- (double) y1*f.b1;
    }
    return f;
}

```

Листинг 7.4. Дополнительные переменные, структуры и функции

В начале обработки удаляются точки, чье смещение было минимальным – поскольку надо «ловить» смещение движущегося объекта. После вычисления формулы линии мы оставляем только вертикальные (или почти вертикальные) линии, т.к. будем отслеживать взмах вверх-вниз.

Посчитав общее количество линий, количество линий направленных вниз и вверх, переходим к этапу определения взмаха. Здесь используется переменная **motion_1**, манипулируя установкой значения которой, можно добиться оптимального по времени взмаха руки.

Если взмах состоялся, то эмулируется нажатие кнопки вправо с помощью функции



`keybd_event()`. Пример слежения за точками представлен на рисунке 7.1.

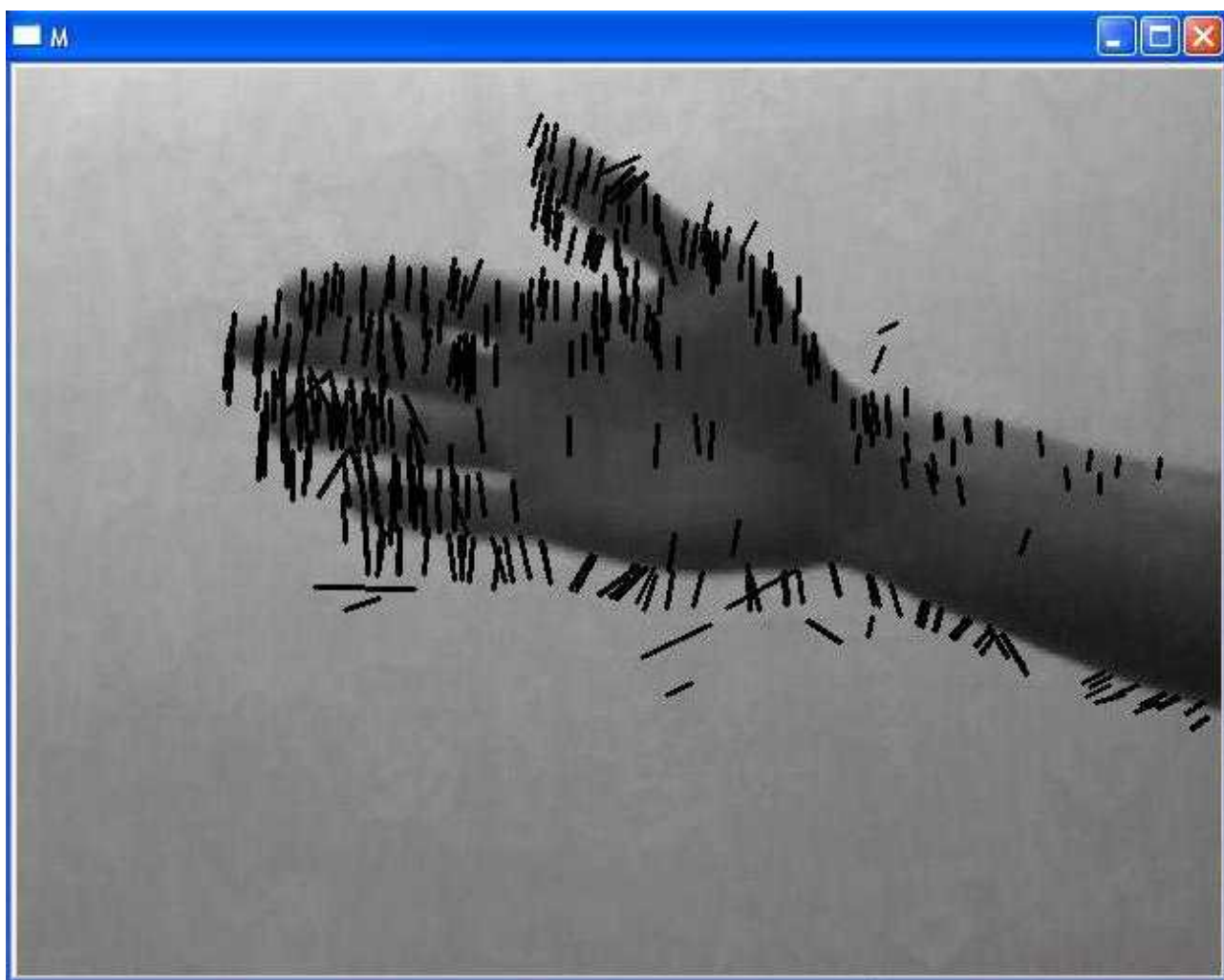


Рис. 7.1 Наблюдение за движением руки

Литература:

1. Bradsky G., Kaehler A. Learning OpenCV - O'Reilly, 2008. – 555 p.
2. <http://cgm.computergraphics.ru/content/view/54>

7.4. Слежение за точками

Пример организации слежения за точками представлен в файле `lkdemo.c` в разделе `samples` (OpenCV). Попробовав запустить пример, вы увидите изображение с предварительно подключённой вами камеры. Нажмите мышкой, к примеру, на вашем носу – появится точка, которая будет отслеживаться компьютером при вашем движении лицом. Программой используется алгоритм Lucas-

Kanade. В листинге 7.5 приведён текст примера с комментариями на русском языке.

```
ifndef _CH_
#pragma package <opencv>
#endif

#define CV_NO_BACKWARD_COMPATIBILITY

#ifndef _EiC
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#include <ctype.h>
#endif

IplImage *image = 0, *grey = 0, *prev_grey = 0, *pyramid = 0, *prev_pyramid =
0, *swap_temp;

int win_size = 10;
const int MAX_COUNT = 500;
CvPoint2D32f* points[2] = {0,0}, *swap_points;
char* status = 0;
int count = 0;
int need_to_init = 0;
int night_mode = 0;
int flags = 0;
int add_remove_pt = 0;
CvPoint pt;

//обработка событий с мыши
void on_mouse( int event, int x, int y, int flags, void* param )
{
    if( !image )
        return;

    if( image->origin )
        y = image->height - y;
```



```

if( event == CV_EVENT_LBUTTONDOWN )
{
    //Если нажали левую кнопку мыши, то добавляем отслеживаемую точку
    pt = cvPoint(x,y);
    add_remove_pt = 1;
}
}

int main( int argc, char** argv )
{
    CvCapture* capture = 0;

    //Получаем видеопоток с камеры или видеофайла, в зависимости от входных
    параметров
    if( argc == 1 || (argc == 2 && strlen(argv[1]) == 1 && isdigit(argv[1][0])) )
        capture = cvCaptureFromCAM( argc == 2 ? argv[1][0] - '0' : 0 );
    else if( argc == 2 )
        capture = cvCaptureFromAVI( argv[1] );

    if( !capture )
    {
        fprintf(stderr, "Could not initialize capturing...\n");
        return -1;
    }

    //Печатается приветствие и краткий хелп по программе
    printf ( "Welcome to lkdemo, using OpenCV version %s (%d.%d.%d)\n",
        CV_VERSION,
        CV_MAJOR_VERSION, CV_MINOR_VERSION, CV_SUBMINOR_VERSION );

    printf( "Hot keys: \n"
        "\tESC - quit the program\n"
        "\tr - auto-initialize tracking\n"
        "\tc - delete all the points\n"

```



```

    "\tn - switch the \"night\" mode on/off\n"
    "To add/remove a feature point click it\n" );

cvNamedWindow( "LkDemo", 0 );

//Устанавливаем функцию обработчик мыши, сама функция находится выше
cvSetMouseCallback( "LkDemo", on_mouse, 0 );

//Главный рабочий цикл программы - работает бесконечно, пока вы
принудительно его не закроете, например нажав ESCAPE
for(;;)
{
    IplImage* frame = 0;
    int i, k, c;

    //Получаем фрейм из видеопотока
    frame = cvQueryFrame( capture );
    if( !frame )
        break;

    if( !image )
    {
        /*Если изображение image не создано, то надо распределить все буферы.
        Изображение не создано первоначально,
        поэтому это всего навсего инициализация.
        */
        image = cvCreateImage( cvGetSize(frame), 8, 3 );
        image->origin = frame->origin;
        grey = cvCreateImage( cvGetSize(frame), 8, 1 );
        prev_grey = cvCreateImage( cvGetSize(frame), 8, 1 );
        pyramid = cvCreateImage( cvGetSize(frame), 8, 1 );
        prev_pyramid = cvCreateImage( cvGetSize(frame), 8, 1 );

        //Выделяем массивы под точки, судя по MAX_COUNT максимально у нас
        может быть 500 точек
        points[0] = (CvPoint2D32f*)cvAlloc(MAX_COUNT*sizeof(points[0][0]));
        points[1] = (CvPoint2D32f*)cvAlloc(MAX_COUNT*sizeof(points[0][0]));
        status = (char*)cvAlloc(MAX_COUNT);
    }
}

```



```

    flags = 0;
}

    //копирование и перевод в чёрно-белое изображение
cvCopy( frame, image, 0 );
cvCvtColor( image, grey, CV_BGR2GRAY );

    //Это для эффекта, когда вы нажимаете n изображение становится тёмным и
вы видите только перемещающиеся
    //назначенные вами точки
if( night_mode )
    cvZero( image );

if( need_to_init )
{
    //Это нужно для автоматической инициализации наблюдения - если вам лень
тыкать мышкой, то можно
    //запустить автоматическое сканирование изображения на факт выделения
подходящих точек

    IplImage* eig = cvCreateImage( cvGetSize(grey), 32, 1 );
    IplImage* temp = cvCreateImage( cvGetSize(grey), 32, 1 );
    double quality = 0.01;
    double min_distance = 10;

    count = MAX_COUNT;

    //определяет наиболее "сильные" углы изображения
    cvGoodFeaturesToTrack( grey, eig, temp, points[1], &count,
        quality, min_distance, 0, 3, 0, 0.04 );

    //уточняет местоположение углов
    cvFindCornerSubPix( grey, points[1], count,
        cvSize(win_size,win_size), cvSize(-1,-1),
        cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03));

    //удаляет ненужные изображения
    cvReleaseImage( &eig );
    cvReleaseImage( &temp );
}

```



```

    add_remove_pt = 0;
}
else if( count > 0 )
{
    //Если есть хотя бы одна точка на экране, то за ней надо следить

    //Вычисляем оптический поток на факт поиска особенностей
    //points[0] - предыдущие точки points[1] - текущие точки
    cvCalcOpticalFlowPyrLK( prev_grey, grey, prev_pyramid, pyramid,
        points[0], points[1], count, cvSize(win_size,win_size), 3, status, 0,
        cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03), flags );
    flags |= CV_LKFLOW_PYR_A_READY;
    //перебираем все точки
    for( i = k = 0; i < count; i++ )
    {
        //Если вы добавили новую точку нажатием мыши и она слишком близка к
        предыдущей - просто удаляем её
        if( add_remove_pt )
        {
            double dx = pt.x - points[1][i].x;
            double dy = pt.y - points[1][i].y;

            if( dx*dx + dy*dy <= 25 )
            {
                add_remove_pt = 0;
                continue;
            }
        }

        if( !status[i] )
            continue;

        points[1][k++] = points[1][i];
        //Отображаем точку на экране
        cvCircle( image, cvPointFrom32f(points[1][i]), 3, CV_RGB(0,255,0), -1,
8,0);

```



```

    }
    count = k;
}

if( add_remove_pt && count < MAX_COUNT )
{
    //Если точка прошла процедуру проверки (см. выше), то добавляем её в
буфер
    points[1][count++] = cvPointTo32f(pt);
    //уточняет местоположение углов для всех точек
    cvFindCornerSubPix( grey, points[1] + count - 1, 1,
        cvSize(win_size, win_size), cvSize(-1, -1),
        cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.03));
    add_remove_pt = 0;
}

CV_SWAP( prev_grey, grey, swap_temp );
CV_SWAP( prev_pyramid, pyramid, swap_temp );
    //Меняем местами точки - новые становятся старыми на следующем шаге цикла
CV_SWAP( points[0], points[1], swap_points );
need_to_init = 0;
cvShowImage( "LkDemo", image );

c = cvWaitKey(10);
if( (char)c == 27 )
    break;
switch( (char) c )
{
case 'r':
    need_to_init = 1;
    break;
case 'c':
    count = 0;
    break;
case 'n':
    night_mode ^= 1;

```



```

        break;
    default:
        ;
    }
}

cvReleaseCapture( &capture );
cvDestroyWindow("LkDemo");

return 0;
}

#ifdef _EiC
main(1, "lkdemo.c");
#endif

```

Листинг 7.5

Итак, запускаем пример и ставим на ручке точку (Рис. 7.2).

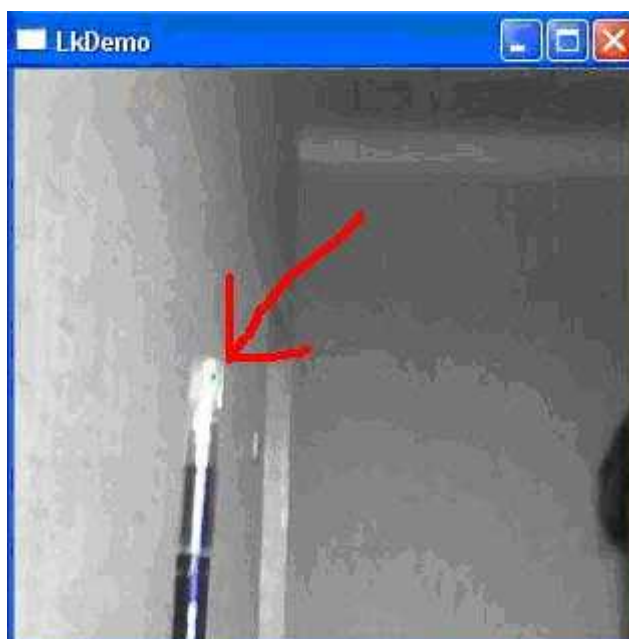


Рис. 7.2

Следим за точкой, перемещая ручку (Рис. 7.3).

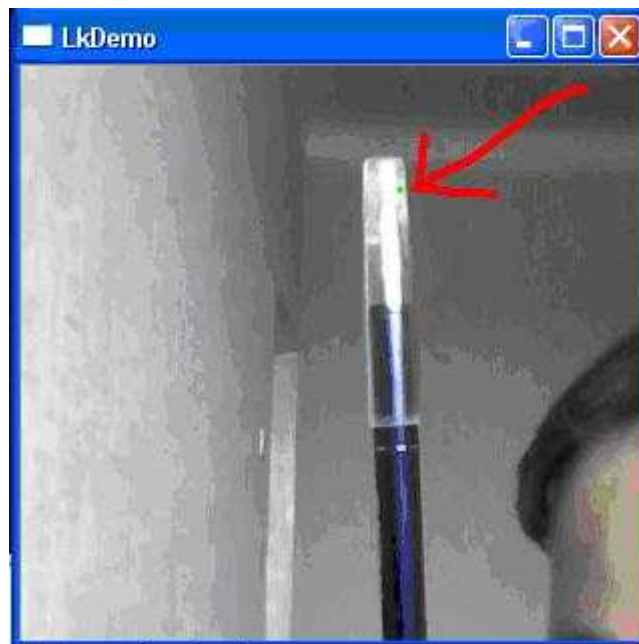


Рис. 7.3

Для пущего эффекта, нажимаем 'n' и смотрим за перемещением точки на чёрном экране (Рис. 7.4).

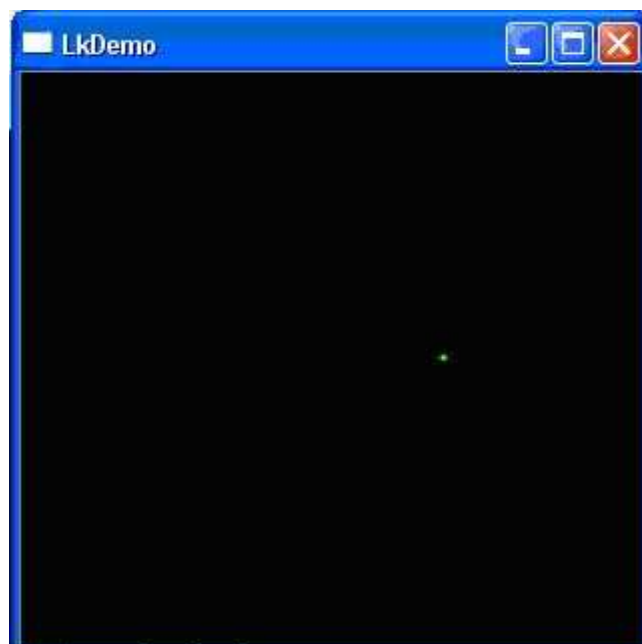


Рис. 7.4

8. СТЕРЕОЗРЕНИЕ

8.1. Стереозрение средствами OpenCV

OpenCV предоставляет для разработчиков функции стереозрения. О теории и внутреннем устройстве поговорим как-нибудь в другой раз, а в этом параграфе просто приведён пример работы с функциями. Для теста использовались картинки, которые используются повсеместно (Рис. 8.1-8.2).



Рис. 8.1. Левое изображение



Рис. 8.2. Правое изображение

В листинге 8.1 представлен программный код позволяющий получить трёхмерные координаты

внутренних объектов.

```
IplImage *image_ = 0;
IplImage *image1_ = 0;
//загрузка тестовых картинок
image_ = cvLoadImage("left.jpg", 1 );
image1_ = cvLoadImage("right.jpg", 1 );
//создание дополнительных изображений
IplImage *image_g = cvCreateImage( cvSize(image_>width,image_>height), 8,
1);
IplImage *image1_g = cvCreateImage( cvSize(image_>width,image_>height), 8,
1);
CvSize size = cvGetSize(image_);
//создание матриц
CvMat* disparity_left = cvCreateMat( size.height, size.width, CV_16S );
CvMat* disparity_right = cvCreateMat( size.height, size.width, CV_16S );
//перевод изображений к градациям серого
cvCvtColor( image_, image_g, CV_BGR2GRAY );
cvCvtColor( image1_, image1_g, CV_BGR2GRAY );
//задание начальных параметров стереозрения
CvStereoGCState* state = cvCreateStereoGCState( 16, 2);
//выполнение основной функции стереозрения, возвращающей матрицу 3d
cvFindStereoCorrespondenceGC( image_g, image1_g, disparity_left,
disparity_right, state, 0 );
cvReleaseStereoGCState( &state );
//вывод матрицы в файл
CvMat* disparity_left_visual = cvCreateMat( size.height, size.width,
CV_8U );
cvConvertScale( disparity_left, disparity_left_visual, -16 );
cvSave( "disparity.png", disparity_left_visual );
cvSaveImage( "disparity.jpg", disparity_left_visual );
return 0;
```

Листинг 8.1. Функции стереозрения на практике

В результате в файле disparity.png будет искомая матрица, а в disparity.jpg – изображение представленное на рисунке 8.3.





Рис. 8.3. Результат работы функции стереозрения

Неинтересно смотреть на плоское изображение. Ни рисунках 8.4-8.5 показаны скриншоты из программы, строящей по матрице disparity.png трёхмерную картину.

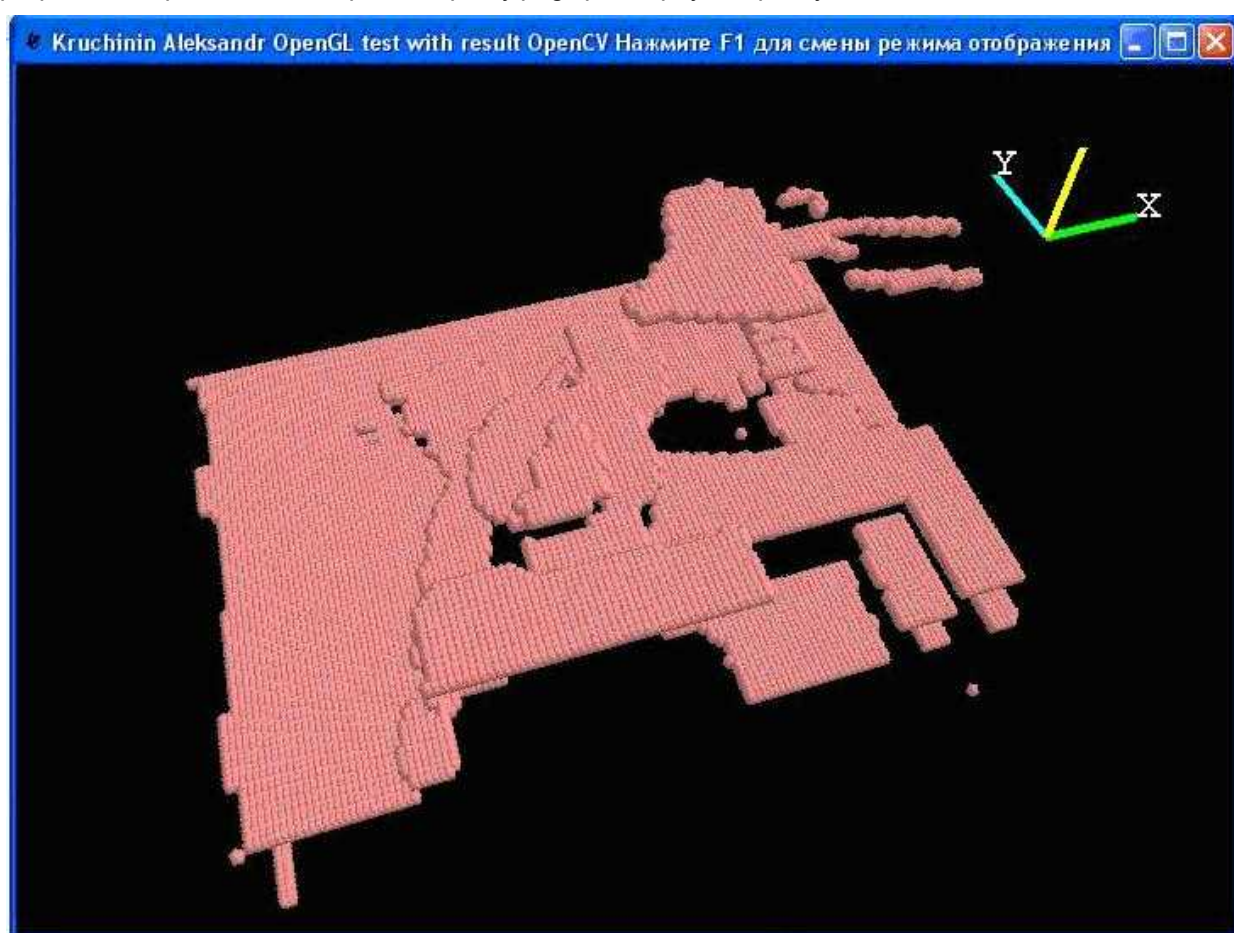


Рис. 8.4. Трёхмерная визуализация без текстур

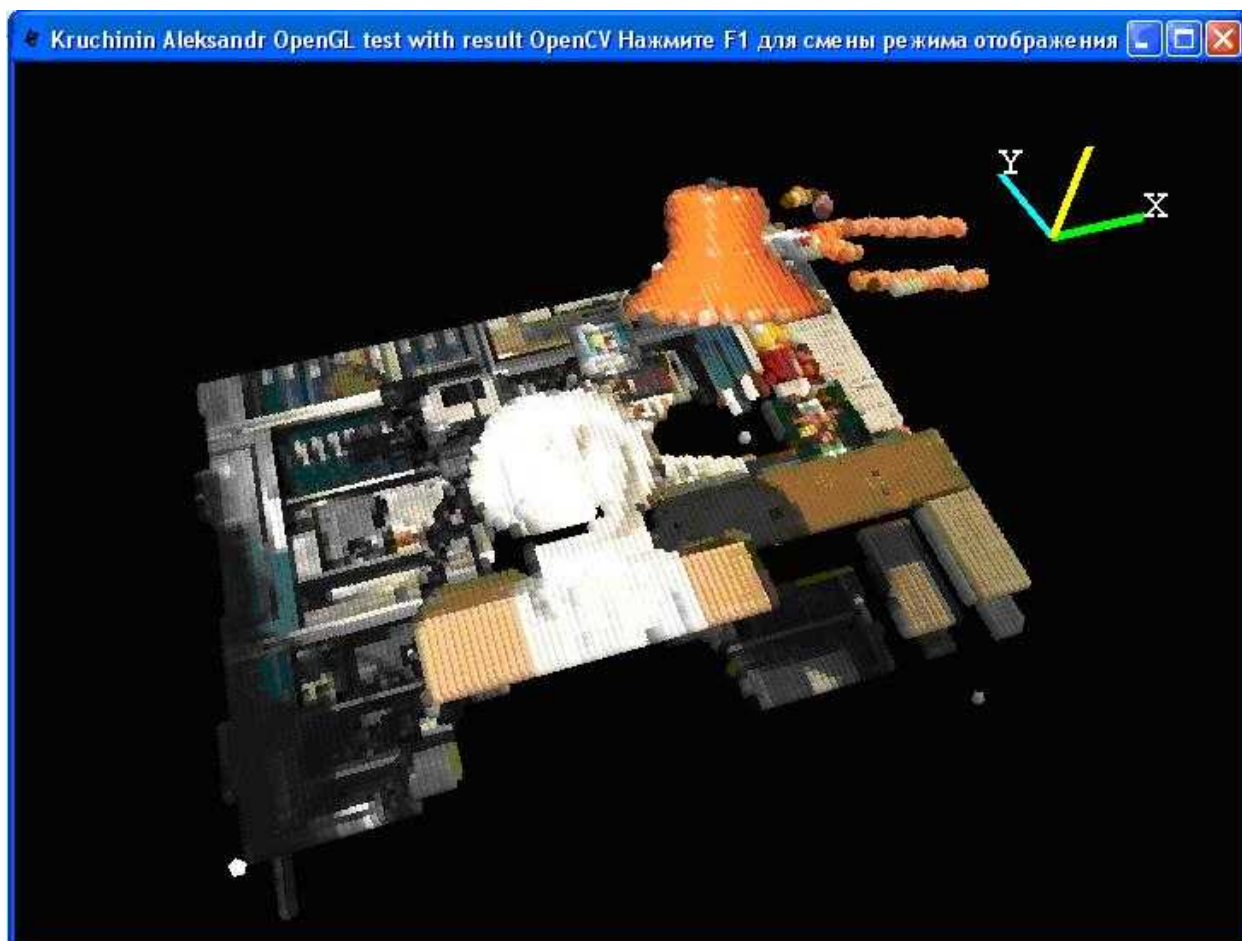


Рис. 8.5. Трёхмерная визуализация с текстурой

По ссылке <http://vidikon.com/download/stereo.zip> вы можете скачать программу, которая и делает это.

8.2. Стереозрение: перевод «различий» в реальные расстояния

Если посмотреть пример, приведённый в листинге предыдущего параграфа, то видно, что в файл записываются не расстояния до объектов, а так называемые «различия». Число 16 в функции `cvCreateStereoGCState` – это общее количество «различий» по глубине (Рис. 8.6 [1]).

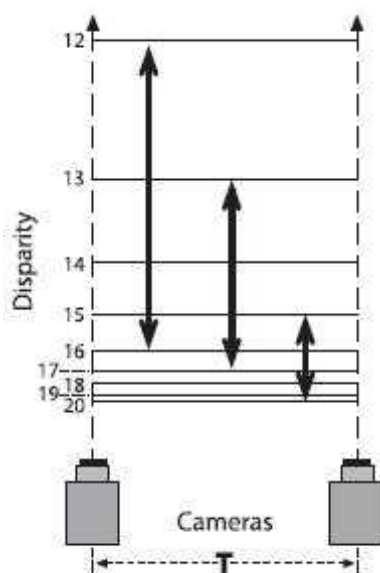


Рис. 8.6. Все точки разбиваются по плоскостям, количество которых задаётся «различиями» – disparity

Для перевода к реальным расстояниям можно воспользоваться функцией **ReprojectImageTo3D**.

ReprojectImageTo3D

```
void cvReprojectImageTo3D(
    const CvArr* disparity,
    CvArr* _3dImage,
    const CvMat* Q
);
```

Параметры:

disparity

Карта различий.

_3dImage

Выходное изображение в 3D формате – 3-канальное, 16-bit integer или 32-bit floating-point.

Q

Матрица перепроектирования 4x4.

Самый интересный – это третий параметр. Как его получить? В документации написано – с помощью функции **cvStereoRectify**. Т.е. после калибровки камеры на выходе получится нужная матрица. А если хочется только попробовать, как эта штука работает? Тогда обращаемся к книге «Learning OpenCV» и смотрим, что представляет из себя матрица:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{bmatrix}$$

Чтобы не быть голословным, объясняя параметры матрицы, обратимся к рисунку 8.7.

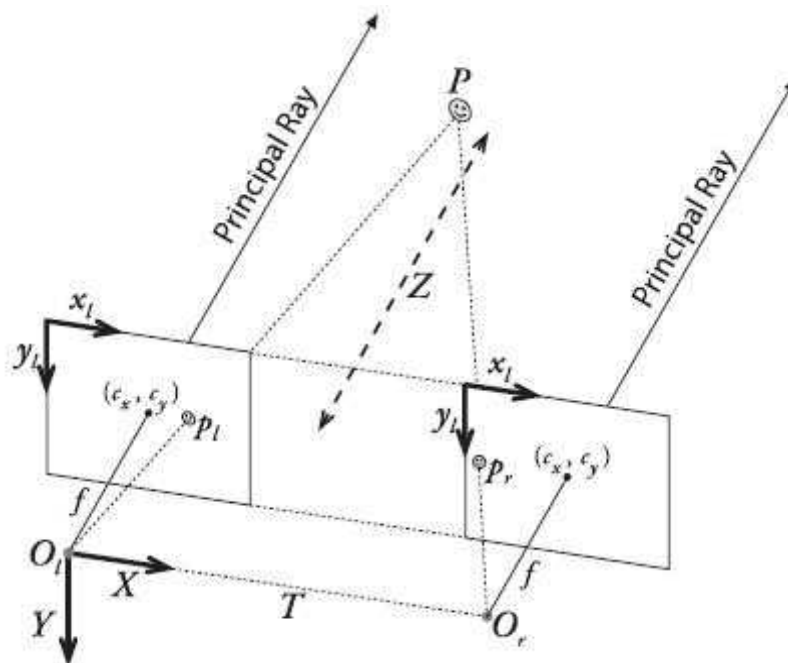


Рис. 8.7. Стере координатная система

T_x – расстояние между камерами по оси X (предполагается, что координаты Y одинаковы); f – фокусное расстояние; c_x, c_y – координаты точки на главном (центрально) луче левой камеры; c'_x – координата на правом изображении левой точки (это делается для того, что уточнить, как связано расстояние между камерами и количество пикселей).

Если всё ещё непонятно, то можете откомпилировать пример из OpenCV по калибровке камер, и посмотреть какие там получаются значения матриц:

```
-          Q          0x0012f17c      double [4][4]
-          [0]        0x0012f17c      double [4]
          [0]        1.0000000000000000      double
          [1]        0.0000000000000000      double
          [2]        0.0000000000000000      double
          [3]        -327.73883438110352      double
-          [1]        0x0012f19c      double [4]
          [0]        0.0000000000000000      double
          [1]        1.0000000000000000      double
          [2]        0.0000000000000000      double
```


	[3]	-239.84486865997314	double
-	[2]	0x0012f1bc	double [4]
	[0]	0.0000000000000000	double
	[1]	0.0000000000000000	double
	[2]	0.0000000000000000	double
	[3]	524.04542174159019	double
-	[3]	0x0012f1dc	double [4]
	[0]	0.0000000000000000	double
	[1]	0.0000000000000000	double
	[2]	-0.30009508961926923	double
	[3]	1.2438668093184739	double

Используя эти значения матрицы, или другие (смоделированные вами), можно вызвать функцию **ReprojectImageTo3D** (Листинг 8.2)

```
IplImage *_3dImage=cvCreateImage( cvSize(image->width,image->height), 32, 3);
cvReprojectImageTo3D(disparity,_3dImage,&_Q);
cvSaveImage( "disparity1.jpg", _3dImage );
cvSave( "disparity1.png", _3dImage );
```

Листинг 8.2. Перевод в 3D формат изображения

На выходе будет массив точек, каждая из которых задаётся координатами x, y, z.

Литература:

1. Bradsky G., Kaehler A. Learning OpenCV - O'Reilly, 2008. – 555 p.

8.3. Функции стереозрения в OpenCV

В OpenCV реализовано два алгоритма стереозрения. Алгоритм Kurt Konolige, являющийся очень быстрым и использующий скользящие суммы абсолютных различий между пикселями в левом изображении и пикселями в правом изображении, и алгоритм вырезки графа FindStereoCorrespondenceGC [1]. С первым алгоритмом связаны функции и структуры, в названиях которых встречается «BM» (block matching – соответствие блока), у второго – «GC».

CvStereoBMState

```
typedef struct CvStereoBMState{
```

```

//Пред-фильтры для нормализации входного изображения:
int    preFilterType; // 0
int    preFilterSize; // ~5x5..21x21
int    preFilterCap; // до ~31
//Соответствие, использующее сумму абсолютного различия
int    SADWindowSize; // Должно быть 5x5..21x21
int    minDisparity; //минимальное различие (=0)
int    numberOfDisparities; // максимальное различие – минимально различие
//Пост-фильтры ,удаляющие плохие значения
int    textureThreshold; // зоны без текстур игнорируются
float  uniquenessRatio;// фильтровать выходные пиксели, если есть закрытые значения
        // с разными различиями
int    speckleWindowSize;// Disparity variation window (НЕ ИСПОЛЬЗУЕТСЯ)
int    speckleRange; // Acceptable range of variation in window (НЕ ИСПОЛЬЗУЕТСЯ)
// Внутренние буферы, не модифицировать
CvMat* preFilteredImg0;
CvMat* preFilteredImg1;
CvMat* slidingSumBuf;
} CvStereoBMState;

```

Изображения сдвинуты на некоторое количество пикселей (между minDisparity и minDisparity+numberOfDisparities). Чтобы улучшать качество алгоритма в него включена пред-фильтрация и пост-фильтрация. Алгоритм работает только при смещении изображений по оси X.

CreateStereoBMState

```

#define CV_STEREO_BM_BASIC 0
#define CV_STEREO_BM_FISH_EYE 1
#define CV_STEREO_BM_NARROW 2

CvStereoBMState* cvCreateStereoBMState(
    int preset=CV_STEREO_BM_BASIC,
    int numberOfDisparities=0
);

```

Параметры:

preset

ID одного из предопределённых наборов параметров. Любые параметры могут быть переопределены после вызова функции.

numberOfDisparities



Число различий. Если 0 – то по умолчанию.

Функция создаёт и инициализирует структуру CvStereoBMState.

ReleaseStereoBMState

```
void cvReleaseStereoBMState(  
    CvStereoBMState** state  
);
```

Параметры:

state

Двойной указатель на освобождаемую структуру.

Функция ReleaseStereoBMState освобождает структуру и все связанные с ней буфера.

FindStereoCorrespondenceBM

```
void cvFindStereoCorrespondenceBM(  
    const CvArr* left,  
    const CvArr* right,  
    CvArr* disparity,  
    CvStereoBMState* state  
);
```

Параметры:

left

Левое одноканальное 8-битное изображение.

right

Правое одноканальное 8-битное изображение.

disparity

Одноканальная выходная 16-битная знаковая карта различий того же размера, что и входные изображения. Её элементы будут содержать различия, умноженные на 16 и округлённые к целому.

state

Сtereo структура CvStereoBMState.

Функция вычисляет карту неравенств по входной паре изображений.

CvStereoGCState

```
typedef struct CvStereoGCState{  
    int lthreshold; // Пороговое значение для функции, анализирующей данные (5 по умолчанию)  
    int interactionRadius; // Радиус размытия (1 по умолчанию; использует модель Поттса)
```



```

float K, lambda, lambda1, lambda2; // Параметры для функции
                                   // (обычно адаптируются к входным образам)

int occlusionCost; // 10000 по умолчанию
int minDisparity; // 0 по умолчанию; см. CvStereoBMState
int numberOfDisparities; // определяется пользователем; см. CvStereoBMState
int maxIters; // количество итераций; определяется пользователем.

// внутренние буферы
CvMat* left;
CvMat* right;
CvMat* dispLeft;
CvMat* dispRight;
CvMat* ptrLeft;
CvMat* ptrRight;
CvMat* vtxBuf;
CvMat* edgeBuf;
} CvStereoGCState;

```

Алгоритм вырезки графа не предназначен для использования в реальном времени. Он даёт очень точную карту глубины с чёткими границами объекта.

CreateStereoGCState

```

CvStereoGCState* cvCreateStereoGCState(
    int numberOfDisparities,
    int maxIters
);

```

Параметры:

numberOfDisparities

Число различий. Диапазон поиска различий: $\text{state} \rightarrow \text{minDisparity} \leq \text{disparity} < \text{state} \rightarrow \text{minDisparity} + \text{state} \rightarrow \text{numberOfDisparities}$

maxIters

Максимальное количество итераций. На каждой итерации испытываются все возможные альфа-расширения. Смотрите [1] для подробностей.

Функция создаёт структуру CvStereoGCState. Её параметры можно поменять вручную после создания.

ReleaseStereoGCState



```
void cvReleaseStereoGCState(  
    CvStereoGCState** state  
);
```

Параметры:

state

Двойной указатель на освобождаемую структуру.

Функция ReleaseStereoGCState освобождает структуру и все связанные с ней буфера.

FindStereoCorrespondenceGC

```
void cvFindStereoCorrespondenceGC(  
    const CvArr* left,  
    const CvArr* right,  
    CvArr* dispLeft,  
    CvArr* dispRight,  
    CvStereoGCState* state,  
    int useDisparityGuess CV_DEFAULT(0)  
);
```

Параметры:

left

Левое одноканальное 8-битное изображение.

right

Правое одноканальное 8-битное изображение.

dispLeft

Необязательная одноканальная левая выходная 16-битная знаковая карта различий того же размера, что и входные изображения.

dispRight

Необязательная одноканальная правая выходная 16-битная знаковая карта различий того же размера, что и входные изображения.

state

Структура CvStereoGCState.

useDisparityGuess

Если параметр не равен 0, алгоритм начинается с предопределённых карт различий. И dispLeft, и dispRight должны содержать правильные карты различий.

Функция cvFindStereoCorrespondenceGC вычисляет карты различий по входной паре изображений. Левое изображение различий будет содержать значения в следующем диапазоне:

$-\text{state->numberOfDisparities} - \text{state->minDisparity} < \text{dispLeft}(x,y) = -\text{state->minDisparity}$,



или

`dispLeft(x,y) == CV_STEREO_GC_OCCLUSION,`

где для правого изображения различий будет верно следующее:

`state->minDisparity = dispRight(x,y) < state->minDisparity+state->numberOfDisparities,`

или

`dispRight(x,y) == CV_STEREO_GC_OCCLUSION,`

т.е. диапазон для левого изображения различий должен быть инверсным, и пиксели, для которых не будут найдены соответствия, будут отмечены как преграды.

Литература:

1. V. Kolmogorov. Graph Based Algorithms for Scene Reconstruction from Two or More Views. PhD thesis, Cornell University, September 2003.



9. МАШИННОЕ ОБУЧЕНИЕ

9.1. Машинное обучение в OpenCV

В OpenCV поддерживаются следующие алгоритмы машинного обучения (Machine Learning).

1. Mahalanobis

Расстояние Махаланобиса. Метод основан на корреляции между переменными, с которой разные модели могут быть выявлены и проанализированы [1,2].

2. K-means

Метод кластеризации k-средних. Алгоритм представляет собой модификацию ЕМ-алгоритма для разделения смеси гауссиан. Он разбивает множество элементов векторного пространства на заранее известное число кластеров k [3,4,5].

3. Normal/Naive Bayes classifier

Порождающий классификатор, особенностью которого является то, что гауссиан распределен и статистически независим друг от друга, что на самом деле не является истиной [6,7].

4. Decision trees

Дерево решений. Дерево находит одну особенность и порог в текущем узле и затем делит данные на отдельные классы. Хотя у данного метода и невысокая достоверность, но он достаточно быстр [8,9].

5. Boosting

Это обучение с учителем. Полное решение классификации основано на весовом комбинировании решений группы классификаторов. При тренировке обучают классификаторы по одному. Каждый классификатор в группе слаб (только чуть-чуть лучше случайного выбора). При тренировке обучается не только решение классификатора, но и какой вес оно будет иметь [10,11].

6. Random trees



Лес отличий из множества деревьев решений, каждое построено для большого или максимального разбиения. В течении обучения каждому узлу в каждом дереве позволяют выбрать глубину переменных только из случайного набора значений данных. Это позволяет гарантировать, что каждое дерево является статистически независимым. Этот алгоритм очень эффективный.

7. Face detector / Haar classifier

Используется boosting. Уже есть обученные классификаторы для распознавания лиц [14].

8. Expectation maximization (EM)

Максимизация ожидания. Порождающий неконтролируемый, используемый для кластеризации. [15,16]

9. K-nearest neighbors

К-ближайших соседей. [17]

10. Neural networks / Multilayer perceptron (MLP)

Больше всего подходит для распознавания символов – медленно обучается, но быстро работает [18].

11. Support vector machine (SVM)

Метод опорных векторов. Это набор схожих алгоритмов вида «обучение с учителем», использующихся для задач классификации и регрессионного анализа. Этот метод принадлежит к семейству линейных классификаторов. Он может также рассматриваться как специальный случай регуляризации по А. Н. Тихонову. Особым свойством метода опорных векторов является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора [19,20].

Литература:

1. P. Mahalanobis, "On the generalized distance in statistics," Proceedings of the National Institute of Science 12 (1936): 49–55.
2. http://en.wikipedia.org/wiki/Mahalanobis_distance
3. S. Lloyd, "Least square quantization in PCM's" (Bell Telephone Laboratories Paper), 1957. ["Lloyd's algorithm" was later published in IEEE Transactions on Information Theory 28 (1982): 129–137.]



4. H. Steinhaus, "Sur la division des corp materiels en parties," Bulletin of the Polish Academy of Sciences and Mathematics 4 (1956): 801–804.
5. <http://ru.wikipedia.org/wiki/K-means>
6. M. Minsky, "Steps toward artificial intelligence," Proceedings of the Institute of Radio Engineers 49 (1961): 8–30.
7. M. E. Maron, "Automatic indexing: An experimental inquiry," Journal of the Association for Computing Machinery 8 (1961): 404–417.
8. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees, Monterey, CA: Wadsworth, 1984.
9. http://en.wikipedia.org/wiki/Decision_tree
10. Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," Journal of Computer and System Sciences 55 (1997): 119–139.
11. <http://en.wikipedia.org/wiki/Boosting>
12. T. K. Ho, "Random decision forest," Proceedings of the 3rd International Conference on Document Analysis and Recognition (pp. 278–282), August 1995.
13. L. Breiman, "Random forests," Machine Learning 45 (2001): 5–32.
14. P. Viola and M. J. Jones, "Robust real-time face detection," International Journal of Computer Vision 57 (2004): 137–154.
15. A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," Journal of the Royal Statistical Society, Series B 39 (1977): 1–38.
16. http://en.wikipedia.org/wiki/Expectation-maximization_algorithm
17. E. Fix, and J. L. Hodges, "Discriminatory analysis, nonparametric discrimination: Consistency properties" (Technical Report 4), USAF School of Aviation Medicine, Randolph Field, Texas, 1951.
18. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in D. E. Rumelhart, J. L. McClelland, and PDP Research Group (Eds.), Parallel Distributed Processing. Explorations in the Microstructures of Cognition (vol. 1, pp. 318–362), Cambridge, MA: MIT Press, 1988.
19. V. Vapnik, The Nature of Statistical Learning Theory, New York: Springer-Verlag, 1995.
20. http://ru.wikipedia.org/wiki/Support_vector_machine

9.2. Нейронные сети в OpenCV (Neural networks)

ML поддерживает нейронные сети прямого распространения, а особенно многослойные перцептроны (MLP). MLP состоит из входного уровня, выходного уровня и одного или нескольких скрытых уровней. Каждый уровень в MLP включает один или более нейронов, который направлено связаны с нейронами предыдущего и следующего уровней. На рисунке 9.1 приведён пример 3-х уровневой перцептрона с 3-мя входными, 2-мя выходными и 5-ю скрытыми нейронами.



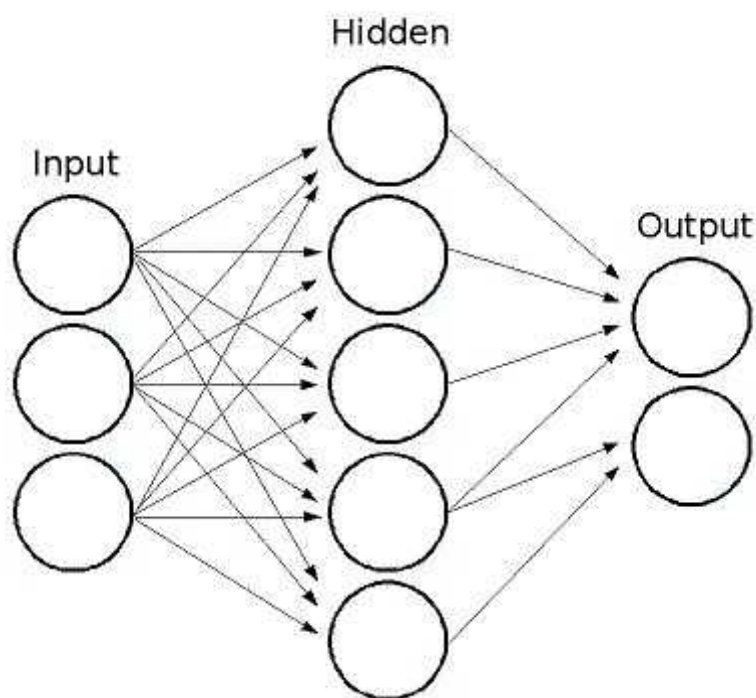


Рис. 9.1. Пример перцептрона

Все нейроны в MLP подобны. Каждый из них имеет несколько входных связей (то есть требуется значения вывода от нескольких нейронов в предыдущем уровне) и нескольких связей вывода (то есть ответ для нескольких нейронов в следующем уровне). Значения, найденные на предыдущем уровне суммированы с некоторыми весовыми коэффициентами, индивидуальными для каждого нейрона, плюс значение смещения (bias), и сумма преобразовывается, используя функцию активации f , которая может быть также различна для различных нейронов (Рис. 9.2).

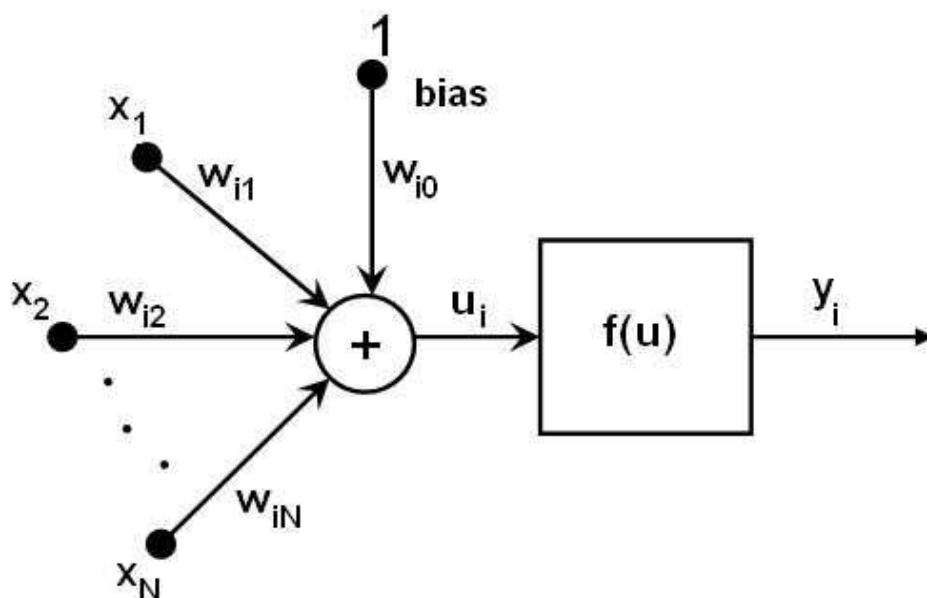


Рис. 9.2

Учитывая выводы $\{x_j\}$ уровня n , выводы $\{y_i\}$ уровня $n+1$ вычислены как:

$$u_i = \sum_j (w_{i,j}^{(n+1)} * x_j) + w_{i,bias}^{(n+1)}$$

$$y_i = f(u_i)$$

Могут использоваться различные функции активации, ML поддерживает три.

1. Функция идентичности (CvANN_MLP::IDENTITY): $f(x)=x$
2. Симметричная сигмовидная (CvANN_MLP::SIGMOID_SYM):

$$f(x) = \beta * (1 - e^{-\alpha x}) / (1 + e^{-\alpha x})$$

Заданный по умолчанию выбор для MLP. Стандартный сигмоид с $\beta=1$, $\alpha=1$ показан на рисунке

9.3.

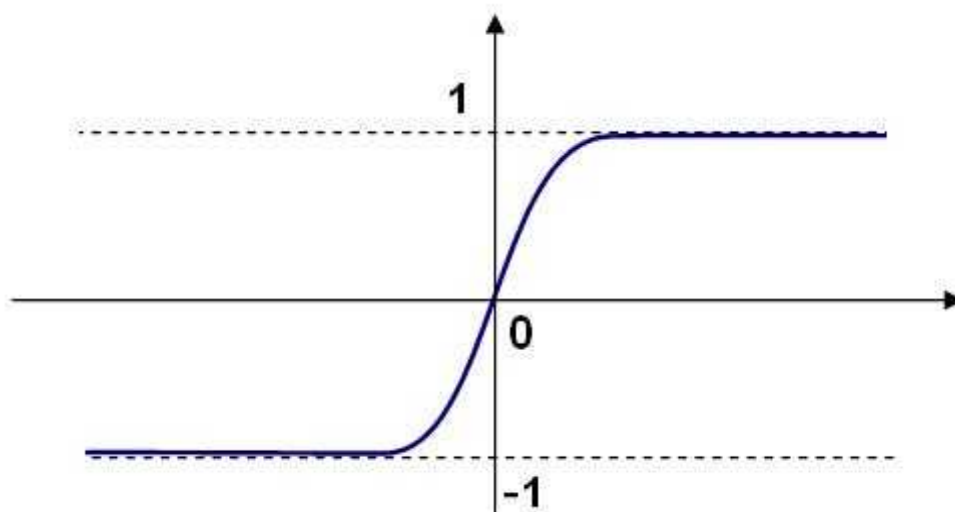


Рис. 9.3. Симметричная сигмовидная функция

3. Функция гауссиана (CvANN_MLP::GAUSSIAN):

$$f(x) = \beta e^{-\alpha x^2}$$

Но она не полностью поддерживается на настоящий момент.

В ML все нейроны имеют те же самые функции активации, с теми же самыми свободными параметрами (α , β), которые определены пользователем и не изменены обучающими алгоритмами.

Для обучения требуется вектор параметров на входе, размер которого соответствует размеру входного уровня, когда значения пропускают до первого скрытого уровня, выводы скрытого уровня вычисляются, используя функции активации, и проходят далее.

Поэтому для вычисления необходимо знать значения весовых коэффициентов. Весовые коэффициенты вычисляются обучающим алгоритмом. Алгоритм обучения включает: многократные входные векторы с соответствующими выходными векторами, и корректировка внутренних весовых коэффициентов для обеспечения максимальной достоверности результата.

Чем больше размер сети (включая скрытые уровни), тем больше гибкость сети, а ошибка минимальна. Однако при увеличении размера сети может возникать ошибка шума, которая после достижения определенного предела начинает расти. Кроме того, большие сети намного дольше обучать, чем малые.

ML поддерживает 2 алгоритма MLP: 1) классический случайный последовательный алгоритм с обратным распространением; 2) пакетный RPROP алгоритм.



10. РАЗНОЕ И ПРАКТИЧЕСКИЕ ПРИМЕРЫ

10. 1. Детектирование лиц и глаз

OpenCV обладает возможностью детектировать лица и глаза на изображениях. Для того, чтобы понять, как это делается, обратимся к следующему примеру (Листинг 10.1). Данный пример модифицирован из стандартного, поставляемого с OpenCV.

```
#include "cv.h"
#include "highgui.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>

#define MAX_EYE 10

static CvMemStorage* storage = 0;
static CvHaarClassifierCascade* cascade = 0;
static CvHaarClassifierCascade* nested_cascade = 0;
int use_nested_cascade = 0;

void detect_and_draw( IplImage* image);

const char* cascade_name =
    "data/haarcascades/haarcascade_frontalface_alt.xml";
const char* nested_cascade_name =
    "data/haarcascades/haarcascade_eye_tree_eyeglasses.xml";
```



```

double scale = 1;

CvSeq* contours = 0;

void Init();

int _tmain(int argc, _TCHAR* argv[])
{
    Init();
    return 0;
}

void Init()
{
    CvCapture* capture = 0;
    IplImage *frame_copy = 0;
    IplImage *image = 0;

    int i;
    const char* input_name = 0;

    //Загрузка базы данных, обученной на детектирование лиц в Фас
    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );
    //Загрузка базы данных, обученной для детектирования глаз
    nested_cascade = (CvHaarClassifierCascade*)cvLoad( nested_cascade_name, 0, 0,
0 );

    if( !cascade )
    {
        return ;
    }

    //Создание хранилища памяти
    storage = cvCreateMemStorage(0);

    char *array_1[4]={

```



```

        "picture03.jpg",
        "image.jpg",
        "26.jpg",
        "27.jpg"
    };

    cvNamedWindow( "result", 1 );

    for(i=0;i<4;i++)
    {
        //Загружаем картинку
        image = cvLoadImage( array_1[i], 1 );
        if( image )
        {
            //Производим детектирование
            detect_and_draw( image);
            //Удаляем картинку
            cvReleaseImage( &image );
        }
    }
}

void detect_and_draw( IplImage* img )
{
    static CvScalar colors[] =
    {
        {{0,0,255}},
        {{0,128,255}},
        {{0,255,255}},
        {{0,255,0}},
        {{255,128,0}},
        {{255,255,0}},
        {{255,0,0}},
        {{255,0,255}}
    };
};

```



```

IplImage *gray, *small_img;
int i, j;

gray = cvCreateImage( cvSize(img->width,img->height), 8, 1 );
small_img = cvCreateImage( cvSize( cvRound (img->width/scale),
                                cvRound (img->height/scale)), 8, 1 );

cvCvtColor( img, gray, CV_BGR2GRAY );
cvResize( gray, small_img, CV_INTER_LINEAR );
cvEqualizeHist( small_img, small_img );
cvClearMemStorage( storage );

if( cascade )
{
    double t = (double)cvGetTickCount();
    CvSeq* faces = cvHaarDetectObjects( small_img, cascade, storage,
                                        1.1, 2, 0
                                        //|CV_HAAR_FIND_BIGGEST_OBJECT
                                        //|CV_HAAR_DO_ROUGH_SEARCH
                                        |CV_HAAR_DO_CANNY_PRUNING
                                        //|CV_HAAR_SCALE_IMAGE
                                        ,
                                        cvSize(30, 30) );

    t = (double)cvGetTickCount() - t;
    printf( "detection time = %gms\n", t/
((double)cvGetTickFrequency()*1000.) );
    for( i = 0; i < (faces ? faces->total : 0); i++ )
    {
        CvRect* r = (CvRect*)cvGetSeqElem( faces, i );
        CvMat small_img_roi;
        CvSeq* nested_objects;
        CvPoint center;
        CvScalar color = colors[i%8];
        int radius;

        center.x = cvRound((r->x + r->width*0.5)*scale);
        center.y = cvRound((r->y + r->height*0.5)*scale);
    }
}

```



```

radius = cvRound((r->width + r->height)*0.25*scale);
cvCircle( img, center, radius, color, 3, 8, 0 );
if( !nested_cascade )
    continue;
cvGetSubRect( small_img, &small_img_roi, *r );
nested_objects = cvHaarDetectObjects( &small_img_roi, nested_cascade,
storage,

    1.1, 2, 0
    //|CV_HAAR_FIND_BIGGEST_OBJECT
    //|CV_HAAR_DO_ROUGH_SEARCH
    //|CV_HAAR_DO_CANNY_PRUNING
    //|CV_HAAR_SCALE_IMAGE
    ,
    cvSize(0, 0) );
for( j = 0; j < (nested_objects ? nested_objects->total : 0); j++ )
{
    CvRect* nr = (CvRect*)cvGetSeqElem( nested_objects, j );
    center.x = cvRound((r->x + nr->x + nr->width*0.5)*scale);
    center.y = cvRound((r->y + nr->y + nr->height*0.5)*scale);
    radius = cvRound((nr->width + nr->height)*0.25*scale);
    cvCircle( img, center, radius, color, 3, 8, 0 );
}
}
}

cvShowImage( "result", img );
cvWaitKey(1000);
cvReleaseImage( &gray );
cvReleaseImage( &small_img );
}

```

Листинг 10.1

Первоначально с помощью функции **cvLoad()** загружаются базы для детектирования лиц и глаз (OpenCV\data\haarcascades\). Вы можете скопировать эти базы в свой каталог или указать на точный путь до OpenCV. Помимо базы для детектирования лиц в фас, есть ещё база для детектирования лиц в профиль. Если базы нормально загружаются, то переходим к созданию хранилища памяти. Затем с помощью функции **cvNamedWindow()** создаём окно, в которое будут выводиться результаты



детектирования, и программа входит цикл в котором последовательно выполняется загрузка изображения в память, детектирование его с использованием функции **detect_and_draw()**, удаление изображения из памяти.

Опишем подробнее процесс детектирования в функции **detect_and_draw()**.

Первоначально определяется массив **colors** – в нём содержатся цвета для окружностей, которыми будут выделяться лица и глаза. С помощью функции **cvCreateImage()** создаются два дополнительных изображения (*IplImage *gray, *small_img;*), на которых и будет осуществляться детектирование. Эти изображения представлены в градациях серого, для перевода в этот формат использовалась функция **cvCvtColor()**. Далее обнуляется хранилище памяти **cvClearMemStorage()**.

С использованием функции **cvHaarDetectObjects()** в последовательность **faces** возвращаются все участки на картинке, которые соответствуют лицу человека. Параметры этой функции вы можете посмотреть в документации OpenCV, но если вкратце, то они характеризуют шаг, через который просматриваются объекты лица, минимальный размер области лица и дополнительные флаги. Посчитав время на детектирование лиц, и выведя его в стандартный поток печати, программа переходит к циклу, где перебираются все участки, соответствующие лицам, которые нашлись.

Первоначально на экран исходное изображение **img** выводится с помощью **cvCircle()** окружность. Затем, если база для детектирования глаз загрузилась успешно, программа с использованием **cvGetSubRect()** устанавливает границы, внутри которых будут искать глаза. А затем, используя ту же функцию **cvHaarDetectObjects()** детектируются глаза внутри области лица. Программа опять входит в цикл, в котором перебираются все найденные глаза и обводятся окружностью.

Если глаз всего один – значит это циклоп, если глаза больше чем три – прищелец. А если серьёзно, то областей глаз может быть больше – и вам придётся выбирать самим, которые реальные, а в которых программа ошиблась.

Используя **cvShowImage()** выводим на экран результат детектирования лица. Ждём секунду (**cvWaitKey(1000)**) и, не забывая удалять созданные картинки, выходим из функции.



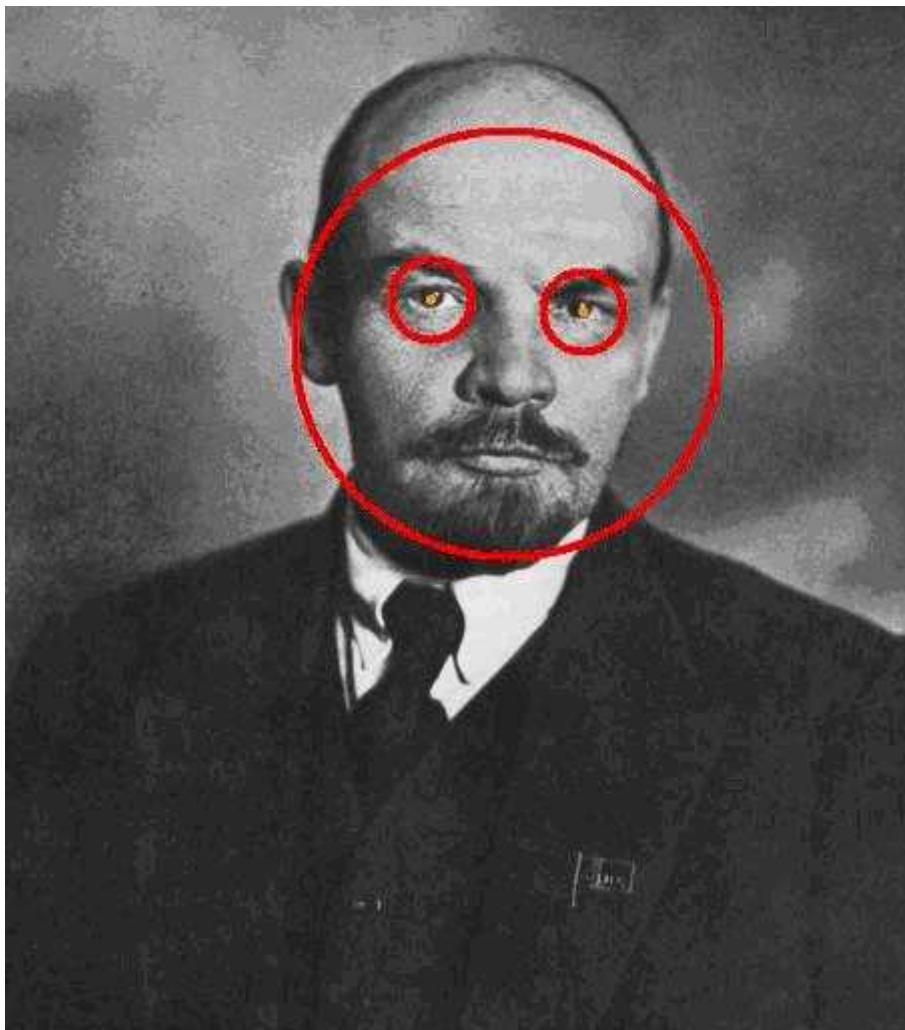


Рис. 10.1. Детектирование лиц и глаз

10.1.1. Можно ли с помощью OpenCV идентифицировать человека по лицу?

После того, как удалось детектировать лицо человека и его глаза на изображении, возникает закономерный вопрос: можно ли идентифицировать человека? К сожалению, встроенных функций идентификации нет. Но вы можете использовать следующий подход.

В различных работах [1, 2] и др. предложены алгоритмы распознавания лиц людей в частности на основе выделения характеристических точек: центров глаз, уголков глаз, кончика носа и т.п. Однако данные методы невозможно использовать в автоматизированной системе распознавания без алгоритма выделения данных точек.

Для нахождения уголков глаз, между ними проводится линия, относительно которой наблюдается изменение яркости. При использовании фотографии, представленной на рисунке

предыдущего параграфа, график изменения яркости между центрами глаз показан на рисунке 10.2.

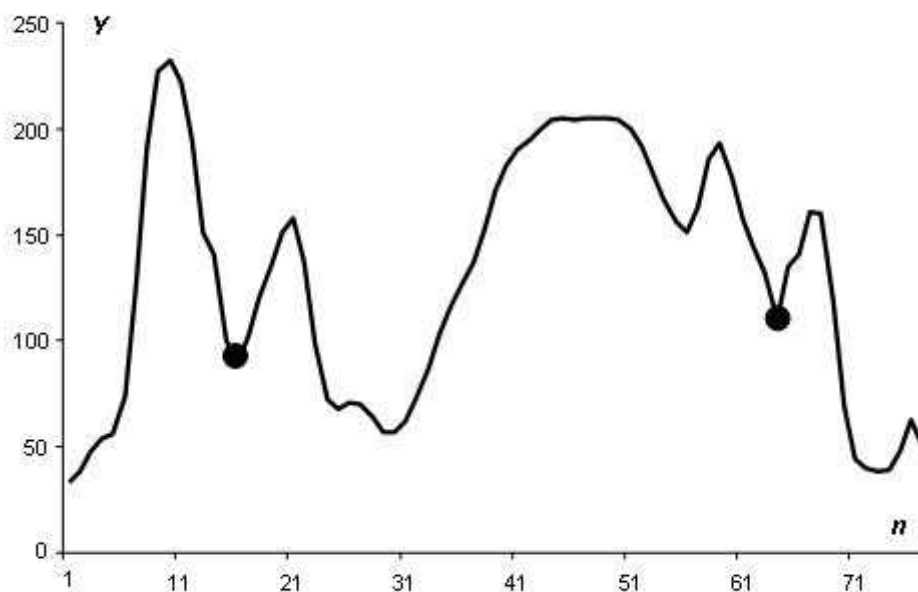


Рис. 10.2. Зависимость яркости Y от номера пикселя n относительно линии начинающейся в центре левого глаза и заканчивающегося в центре правого

По левому и правому краям графика находятся минимумы, соответствующие зрачкам глаз. Резкое возрастание яркости означает переход к белку глаза. Последующее уменьшение яркости соответствует уголку глаза.

При нахождении кончика носа необходимо найти центр между глазами, от которого построить перпендикуляр в нижнюю сторону лица. После чего также построить зависимость яркости (Рис. 10.3).

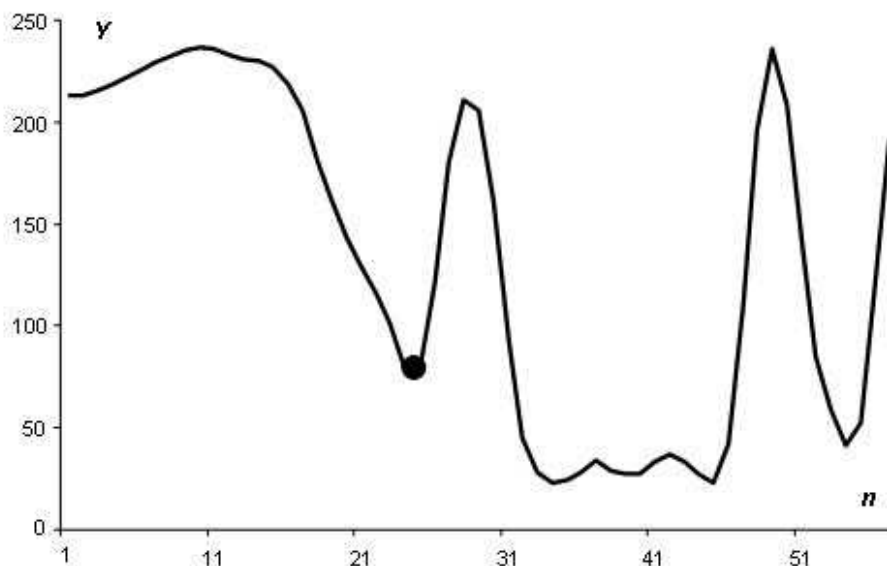


Рис. 10.3. Зависимость яркости Y от номера пикселя n относительно линии для определения кончика носа

Как видно из рисунка 10.3, первое значительное снижения яркости означает искомый кончик носа, затем усы и другие части лица. В результате найдены три значимые характеристические точки (рис. 10.4). Аналогичным образом можно найти и любые другие точки.

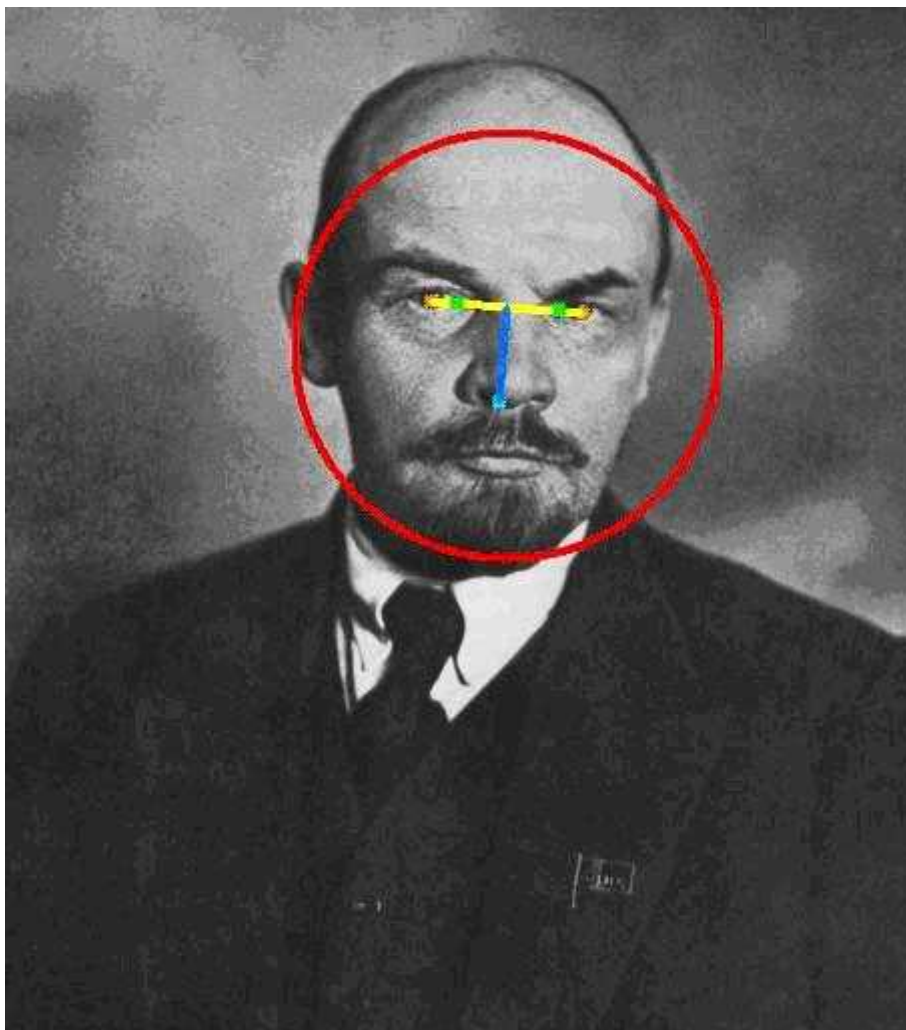


Рис. 10.4. Найденные характеристические точки

Предложенный подход определения характеристических точек по изменению яркости позволяет в автоматизированном режиме, например при анализе потока видеоинформации с Web-камеры, осуществлять идентификацию лиц. Недостатком данного подхода является то, что функция `cvHaarDetectObjects` недостаточно точно функционирует на большинстве изображениях. Поэтому целесообразность использования библиотеки OpenCV для решения задачи идентификации лиц вызывает сомнения. Хотя на начальном этапе, при проведении исследований, OpenCV безусловно вам пригодится.

1. Wiskott L., Fellous J.-M., Krueger N and Malsburg C. Face Recognition by Elastic Bunch Graph Matching. IEEE Transactions on Pattern Analysis and Machine Intelligence 1997, Vol. 19, pp. 775-779.

10.1.2. Реализация трекинга лица

Алгоритм использует следующие технологии:

- 1) детектируется лицо ;
- 2) на лице выделяются точки для трекинга и затем осуществляется их мониторинг .

Я не стану приводить код и того и другого, поскольку используются вещи из стандартных примеров, описанных в частности ранее. После детектирования лица нам известен `CvRect`, который передаём в функцию `cvSetImageROI` для текущего изображения с Web-камеры. Ниже приведён пример инициализации точек:

```
IplImage* eig = cvCreateImage( cvGetSize(grey), 32, 1 );
IplImage* temp = cvCreateImage( cvGetSize(grey), 32, 1 );
double quality = 0.01;
double min_distance = 10;
cvSetImageROI(grey, RectFace);
count = MAX_COUNT;
cvGoodFeaturesToTrack( grey, eig, temp, points[1], &count,
    quality, min_distance, 0, 3, 0, 0.04 );
cvFindCornerSubPix( grey, points[1], count,
    cvSize(win_size, win_size), cvSize(-1, -1),
    cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20, 0.03));
cvResetImageROI(grey);
```

Листинг 10.2. Инициализация точек трекинга

Если какие-то параметры непонятные, то посмотрите пример OpenCV по трекингу – я их имена не менял. Но возникает резонный вопрос: а зачем вообще использовать трекинг, если можно постоянно детектировать лицо? Ответ: без использования пакета распараллеливания процедура детектирования лица полностью занимает ресурсы современного персонального компьютера, причём так, что не успевают обрабатываться все кадры. А при трекинге – на средненьком компьютере Core 2 Duo 2.2 ГГц затрачиваются не более 5% ресурсов центрального процессора.

Для определения положения лица и его наклона будем оперировать двумя переменными: `Point1` и `Point2`. В следующем листинге показано их начальное определение после детектирования лица:

```
Point1.x=0;
```



```

Point1.y=0;
Point2.x=0;
Point2.y=0;
int pp=0;
for (i=0;i<count;i++) {
    points[1][i].x+=RectFace.x;
    points[1][i].y+=RectFace.y;
    //Здесь также необходимо считать средние точки
    Point1.x+=points[1][i].x;
    Point1.y+=points[1][i].y;
    point_status[i]=0;
    if (points[1][i].y<CenterFace.y)
    {
        point_status[i]=1;
        Point2.x+=points[1][i].x;
        Point2.y+=points[1][i].y;
        pp++;
    }
}
Point1.x/=count;
Point1.y/=count;
Point2.x/=pp;
Point2.y/=pp;

```

Листинг 10.3. Определение точек Point1 и Point2

Первоначально точки обнуляются. Затем перебираются все точки трекинга, к которым прибавляется смещение CvRect лица – RectFace. В Point1 суммируются все точки – эта точка будет нам показывать, где центр лица. Для определения наклона используется довольно простое соображение – первоначально считается, что лицо расположено прямо (можно было в принципе сразу определить наклон по детектированным глазам), и выделяются все точки, которые выше центра лица – CenterFace. Этим точкам устанавливается специальный статус. После завершения цикла вычисляются средние значения точек. Собственно, проведя линию между Point1 и Point2, мы получаем наклон оси.

При осуществлении трекинга необходимо обновлять значения точек Point1 и Point2, точка Point1 – это средняя всех точек, а точка Point2 – это средняя точка всех точек со статусом point_status.

Для вывода другого лица поверх своего (у нас это лицо вождя мирового пролетариата) необходимо получение координаты точки вывода и наклона лица. В листинге 3 показано, как это



делается.

```
double angle;
if (CenterFace.y-CenterE.y!=0) angle=atan((double) (CenterFace.x-CenterE.x) /
(CenterFace.y-CenterE.y));
else angle=0;
double angle_=angle*180/PI;
CvSize Size1=cvGetSize(image_1);
//ещё угол
double angle1=atan((double)Size1.width/Size1.height);
Mat image_11=image_1;
Mat image_12=rotateImage(image_11,angle_);
IplImage image_2=image_12;
CvSize Size=cvGetSize(&image_2);
CvRect Rect;
double x=Size.width;
double y=Size.height;
Rect.x=CenterFace.x-x/2;Rect.y=CenterFace.y-y/2;
Rect.height=Size.height;
Rect.width=Size.width;
cvSetImageROI(image,Rect);
IplImage* image_2m=cvCreateImage(cvGetSize(&image_2), 8, 1);
cvCvtColor( &image_2,image_2m, CV_BGR2GRAY );
if (Rect.x>=0 && Rect.y>=0 && Rect.x+Rect.width<=639 &&
Rect.y+Rect.height<=479)
    cvCopy(&image_2,image,image_2m);
cvResetImageROI( image);
cvReleaseImage(&image_2m);
```

Листинг 10.4. Вычисление координаты, угла и вывод изображения на экран

Первоначально вычисляется угол наклона angle, CenterFace до этого выражается через Point1, а CenterE – через Point2. Вычисляется размер изображения Вождя - Size1. Далее происходит поворот изображения и установление прозрачного цвета . Ну и собственно вывод изображения на экран.

Конечно, это только подход, и немного оптимизировав его можно добиться более серьёзных результатов. Но на это надо уже больше времени.



10.2. Особенности и функция ExtractSURF

Эта функция предназначена для выделения особенностей на изображении, которые могут использоваться для сравнения объектов.

Функция ExtractSURF

```
void cvExtractSURF(  
    const CvArr* image,  
    const CvArr* mask,  
    CvSeq** keypoints,  
    CvSeq** descriptors,  
    CvMemStorage* storage,  
    CvSURFParams params  
);
```

Параметры:

image

Входное 8-битное изображение (градации серого).

mask

Необязательная входная 8-битная маска. Если маска не нужна, то этот параметр равен 0. Особенности находятся только в тех областях, которые содержат больше чем 50% пикселей маски неравных нулю.

keypoints

Выходной параметр; двойной указатель на последовательность ключевых точек. Это будет последовательность структуры CvSURFPoint.

descriptors

Необязательный параметр вывода; двойной указатель на последовательность описателей; в зависимости от значения params.extended, каждый элемент последовательности будет или с 128 элементами с плавающей запятой (CV_32F) или вектор с 64 элементами. Если параметр NULL, описатели не вычисляются.

storage

Хранилище памяти для последовательности точек и дескрипторов.

params

Различные параметры алгоритма объединены в структуре CvSURFParams.

Функция cvExtractSURF находит хорошие особенности изображения так, как описано в [1]. Для каждой особенности возвращает ее местоположение, размер, ориентацию и произвольный описатель. Функция может использоваться для прослеживания объекта и локализации, изображения и т.д.



Структура CvSURFPoint

```
typedef struct CvSURFPoint{
    CvPoint2D32f pt; // позиция особенности в пределах изображения
    int laplacian; // -1, 0 или +1. Признак лапласиана в точке.
    // Может использоваться для ускорения сравнения особенностей
    // (обычно особенности с лапласианами не могут соответствовать)
    int size; // размер особенности
    float dir; // ориентация особенности: угол 0..360
    float hessian; // значение матрицы Гессе (может использоваться, чтобы
        //приблизительно оценить силы особенности;
        // подробнее в params.hessianThreshold)
}CvSURFPoint;
```

Структура CvSURFParams

```
typedef struct CvSURFParams{
    int extended; // Если 0, то стандартные дескрипторы (64 элемента каждый),
        // 1 – расширенные дескрипторы (128 элементов каждый)
    double hessianThreshold; // Только особенности с keypoint.hessian большим, чем должны быть
        извлечены
    // Хорошее заданное по умолчанию значение - ~300-500 (может зависеть от среднего числа
    // локального контраста и точность изображения).
    // Пользователь может далее отфильтровывать некоторые особенности, основанные на их
    значениях матрицы Гессе
    // или других характеристик
    int nOctaves; // число октав, которые нужно использовать для извлечения.
        // С каждой следующей октавой размер особенности удвоен (3 по умолчанию)
    int nOctaveLayers; // номер уровней в пределах каждой октавы (4 по умолчанию)
}CvSURFParams;
```

CvSURFParams cvSURFParams(double hessianThreshold, int extended=0); //возвращает значения по умолчанию

Литература:

1. Herbert Bay, Tinne Tuytelaars and Luc Van Gool "SURF: Speeded Up Robust Features", Proceedings of the 9th European Conference on Computer Vision, Springer LNCS volume 3951, part 1, pp 404–417, 2006.



10.2.1. Нахождение объектов на изображении с использованием особенностей

Описанный в предыдущем параграфе подход к определению особенностей можно применять для нахождения объектов на изображении. Есть пример использования в OpenCV – find_obj.cpp. Есть объект (Рис. 10.5) и сцена (Рис. 10.6).



Рис. 10.5. Объект распознавания



Рис. 10.6. Сцена с искомым объектом

Ниже представлен текст программы примера из OpenCV с комментариями (Листинг 10.5).

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

#include <iostream>
#include <vector>

using namespace std;

IplImage *image = 0;

// Сравнение двух особенностей
double
compareSURFDescriptors( const float* d1, const float* d2, double best, int
length )
{
    double total_cost = 0;
    assert( length % 4 == 0 );
    for( int i = 0; i < length; i += 4 )
    {
        double t0 = d1[i] - d2[i];
        double t1 = d1[i+1] - d2[i+1];
        double t2 = d1[i+2] - d2[i+2];
        double t3 = d1[i+3] - d2[i+3];
        total_cost += t0*t0 + t1*t1 + t2*t2 + t3*t3;
        if( total_cost > best )
            break;
    }
    return total_cost;
}

// Сравнивает одну особенность объекта со всеми особенностями сцены
int
naiveNearestNeighbor( const float* vec, int laplacian,
```



```

        const CvSeq* model_keypoints,
        const CvSeq* model_descriptors )
{
    int length = (int) (model_descriptors->elem_size/sizeof(float));
    int i, neighbor = -1;
    double d, dist1 = 1e6, dist2 = 1e6;
    CvSeqReader reader, kreader;
    // Начальная особенность сцены
    cvStartReadSeq( model_keypoints, &kreader, 0 );
    cvStartReadSeq( model_descriptors, &reader, 0 );

    // Перебор всех особенностей сцены
    for( i = 0; i < model_descriptors->total; i++ )
    {
        const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
        const float* mvec = (const float*)reader.ptr;
        CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
        CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
        // Для ускорения сначала сравнивается лапласиан особенностей
        if( laplacian != kp->laplacian )
            continue;
        // Сравнение особенностей
        d = compareSURFDescriptors( vec, mvec, dist2, length );
        if( d < dist1 )
        {
            // Найдена лучшее совпадение особенностей
            dist2 = dist1;
            dist1 = d;
            neighbor = i;
        }
        else if ( d < dist2 )
            dist2 = d;
    }
    if ( dist1 < 0.6*dist2 )
        return neighbor;
    return -1;
}

```



```

}

// Функция ищет совпадающие пары
void
findPairs( const CvSeq* objectKeypoints, const CvSeq* objectDescriptors,
           const CvSeq* imageKeypoints, const CvSeq* imageDescriptors, vector<int>&
ptpairs )
{
    int i;
    CvSeqReader reader, kreader;
    // Установка начальной особенности объекта распознавания
    cvStartReadSeq( objectKeypoints, &kreader );
    cvStartReadSeq( objectDescriptors, &reader );
    ptpairs.clear();

    // Перебор всех особенностей объекта
    for( i = 0; i < objectDescriptors->total; i++ )
    {
        const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
        const float* descriptor = (const float*)reader.ptr;
        CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
        CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
        // Сравнение текущей особенности со всеми особенностями из сцены
        int nearest_neighbor = naiveNearestNeighbor( descriptor, kp->laplacian,
imageKeypoints, imageDescriptors );
        if( nearest_neighbor >= 0 )
        {
            // Нашлось совпадение особенностей
            ptpairs.push_back(i);
            ptpairs.push_back(nearest_neighbor);
        }
    }
}

/* Грубое нахождение местоположения объекта */
int
locatePlanarObject( const CvSeq* objectKeypoints, const CvSeq*

```



```

objectDescriptors,
    const CvSeq* imageKeypoints, const CvSeq* imageDescriptors,
    const CvPoint src_corners[4], CvPoint dst_corners[4] )
{
    double h[9];
    CvMat _h = cvMat(3, 3, CV_64F, h);
    vector<int> ptpairs;
    vector<CvPoint2D32f> pt1, pt2;
    CvMat _pt1, _pt2;
    int i, n;
    // Ищем пары особенностей на обеих картинках, которые соответствуют
    // друг другу
    findPairs( objectKeypoints, objectDescriptors, imageKeypoints,
imageDescriptors, ptpairs );
    n = ptpairs.size()/2;
    // Если пар мало, значит надо выходить - объект не найден
    if( n < 4 )
        return 0;
    // Выделяем память
    pt1.resize(n);
    pt2.resize(n);
    // Считываем координаты «особых»точек
    for( i = 0; i < n; i++ )
    {
        pt1[i] = ((CvSURFPoint*)cvGetSeqElem(objectKeypoints,ptpairs[i*2]))->pt;
        pt2[i] = ((CvSURFPoint*)cvGetSeqElem(imageKeypoints,ptpairs[i*2+1]))->pt;
    }

    // По полученным векторам создаём матриц
    _pt1 = cvMat(1, n, CV_32FC2, &pt1[0] );
    _pt2 = cvMat(1, n, CV_32FC2, &pt2[0] );
    // Находим трансформацию между исходным изображением и с тем, которое
    // ищем
    if( !cvFindHomography( &_pt1, &_pt2, &_h, CV_RANSAC, 5 ))
        return 0;
    // По полученному значению трансформации (в матрицу _h) находим

```



```

// координаты четырёхугольника, характеризующего объект
for( i = 0; i < 4; i++ )
{
    double x = src_corners[i].x, y = src_corners[i].y;
    double Z = 1./(h[6]*x + h[7]*y + h[8]);
    double X = (h[0]*x + h[1]*y + h[2])*Z;
    double Y = (h[3]*x + h[4]*y + h[5])*Z;
    dst_corners[i] = cvPoint(cvRound(X), cvRound(Y));
}

return 1;
}

int main(int argc, char** argv)
{
    // Инициализация параметров
    const char* object_filename = argc == 3 ? argv[1] : "box.png";
    const char* scene_filename = argc == 3 ? argv[2] : "box_in_scene.png";

    CvMemStorage* storage = cvCreateMemStorage(0);

    cvNamedWindow("Object", 1);
    cvNamedWindow("Object Correspond", 1);

    static CvScalar colors[] =
    {
        {{0,0,255}},
        {{0,128,255}},
        {{0,255,255}},
        {{0,255,0}},
        {{255,128,0}},
        {{255,255,0}},
        {{255,0,0}},
        {{255,0,255}},
        {{255,255,255}}
    };
};

```



```

// Загрузка изображений
IplImage* object = cvLoadImage( object_filename, CV_LOAD_IMAGE_GRAYSCALE );
IplImage* image = cvLoadImage( scene_filename, CV_LOAD_IMAGE_GRAYSCALE );
if( !object || !image )
{
    fprintf( stderr, "Can not load %s and/or %s\n",
        "Usage: find_obj [<object_filename> <scene_filename>]\n",
        object_filename, scene_filename );
    exit(-1);
}

// Перевод в градации серого
IplImage* object_color = cvCreateImage(cvGetSize(object), 8, 3);
cvCvtColor( object, object_color, CV_GRAY2BGR );

CvSeq *objectKeypoints = 0, *objectDescriptors = 0;
CvSeq *imageKeypoints = 0, *imageDescriptors = 0;
int i;

// Инициализация структуры CvSURFParams с размером дескрипторов в 128
// элементов
CvSURFParams params = cvSURFParams(500, 1);

// Засекаем время
double tt = (double)cvGetTickCount();

// Ищем особенности объекта распознавания
cvExtractSURF( object, 0, &objectKeypoints, &objectDescriptors, storage,
params );

printf("Object Descriptors: %d\n", objectDescriptors->total);

// Ищем особенности сцены
cvExtractSURF( image, 0, &imageKeypoints, &imageDescriptors, storage,
params );

printf("Image Descriptors: %d\n", imageDescriptors->total);

// Сколько потребовалось времени (У меня 167 милли секунд)
tt = (double)cvGetTickCount() - tt;

printf( "Extraction time = %gms\n", tt/(cvGetTickFrequency()*1000.));

// Устанавливаем границы изображений, внутри которых будут сравниваться

```



```

//особенности

CvPoint src_corners[4] = {{0,0}, {object->width,0}, {object->width, object-
>height}, {0, object->height}};

CvPoint dst_corners[4];

// Создание дополнительного изображение (в нём будет сцена и объект)
// Запустите пример и поймёте о чём речь

IplImage* correspond = cvCreateImage( cvSize(image->width, object-
>height+image->height), 8, 1 );

cvSetImageROI( correspond, cvRect( 0, 0, object->width, object->height ) );
cvCopy( object, correspond );

cvSetImageROI( correspond, cvRect( 0, object->height, correspond->width,
correspond->height ) );

cvCopy( image, correspond );
cvResetImageROI( correspond );

// Вызываем функцию, находящую объект на экране
if( locatePlanarObject( objectKeypoints, objectDescriptors, imageKeypoints,
imageDescriptors, src_corners, dst_corners ) )
{
    // Обводим нужный четырёхугольник

    for( i = 0; i < 4; i++ )
    {
        CvPoint r1 = dst_corners[i%4];
        CvPoint r2 = dst_corners[(i+1)%4];
        cvLine( correspond, cvPoint(r1.x, r1.y+object->height ),
            cvPoint(r2.x, r2.y+object->height ), colors[8] );
    }
}

// Если в этом месте вывести результат на экран, то получится то, что
// показано на рисунке 23.3.

vector<int> ptpairs;

// Снова ищутся все совпадающие пары особенностей в обеих картинках
findPairs( objectKeypoints, objectDescriptors, imageKeypoints,
imageDescriptors, ptpairs );

// Между парами особенностей на рисунке проводятся прямые

```




```

for( i = 0; i < (int)ptpairs.size(); i += 2 )
{
    CvSURFPoint* r1 = (CvSURFPoint*)cvGetSeqElem( objectKeypoints,
ptpairs[i] );
    CvSURFPoint* r2 = (CvSURFPoint*)cvGetSeqElem( imageKeypoints,
ptpairs[i+1] );
    cvLine( correspond, cvPointFrom32f(r1->pt),
        cvPoint(cvRound(r2->pt.x), cvRound(r2->pt.y+object->height)),
colors[8] );
}

// Результат можно посмотреть на рисунке 23.4.
cvShowImage( "Object Correspond", correspond );

// Выделяем особенности окружностями (Рис. 23.5)
for( i = 0; i < objectKeypoints->total; i++ )
{
    CvSURFPoint* r = (CvSURFPoint*)cvGetSeqElem( objectKeypoints, i );
    CvPoint center;
    int radius;
    center.x = cvRound(r->pt.x);
    center.y = cvRound(r->pt.y);
    radius = cvRound(r->size*1.2/9.*2);
    cvCircle( object_color, center, radius, colors[0], 1, 8, 0 );
}
cvShowImage( "Object", object_color );

cvWaitKey(0);

cvDestroyWindow("Object");
cvDestroyWindow("Object SURF");
cvDestroyWindow("Object Correspond");

return 0;
}

```

Листинг 10.5. Поиск объектов на сцене





Рис. 10.7. Выделен искомый объект

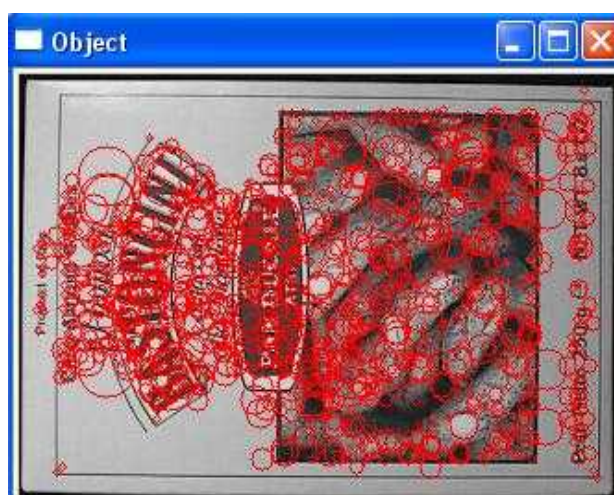


Рис. 10.8. Все особенности объекта

10.2.2. Использование особенностей для распознавания образов

Ранее показано как использовать особенности для нахождения объектов в изображении. Попробуем протестировать данный подход при работе с базами изображений для выяснения достоверности. Для тестирования можно использовать следующую базу образов:

<http://staff.science.uva.nl/~aloi/> .

Поскольку OpenCV использует особенности для изображения в градациях серого, я скачал файл aloi_grey_red2_col.tar. Для тестирования была написана простенькая программа (Листинг 10.6).

```
//Загружаем и обучаем
int i;
char buf[256];
CvSURFParams params = cvSURFParams(500, 1);
CvSeq *objectKeypoints[1000], *objectDescriptors [1000];
CvMemStorage* storage = cvCreateMemStorage(0);
for(i=0;i<500;i++)
{
    sprintf(buf, "e:\\algor\\grey2\\%d\\%d_i110.png", i+1, i+1);
    IplImage* object=cvLoadImage(buf, CV_LOAD_IMAGE_GRAYSCALE);
    IplImage* object_color = cvCreateImage(cvGetSize(object), 8, 3);
    cvCvtColor( object, object_color, CV_GRAY2BGR );
    objectKeypoints[i]=0;objectDescriptors[i]=0;
    double tt = (double)cvGetTickCount();
    cvExtractSURF( object, 0, &objectKeypoints[i], &objectDescriptors[i],
storage, params );
    printf("%d Image Descriptors: %d\n", i+1, objectDescriptors[i]->total);
    tt = (double)cvGetTickCount() - tt;
    printf( "Extraction time = %gms\n", tt/(cvGetTickFrequency()*1000.));
    cvReleaseImage(&object);
    cvReleaseImage(&object_color);
}

LOG_FILE log;

log.BLog("out1.log");
```



```

//А теперь необходимо произвести сравнение
int j,k;
int all=0;
int all_p=0;
int pai_r[1000];
int maxpair;
int il;
for(i=0;i<500;i++)
{
    sprintf(buf,"Begin %d-s image:",i+1);
    log.Log(buf);
    for(j=0;j<11;j++)
    {
        k=120+j*10;
        if (j>=8) k=210+(j-8)*20;
        sprintf(buf,"e:\\algor\\grey2\\%d\\%d_i%d.png",i+1,i+1,k);
        log.Log(buf);
        CvSeq *objectKeypoints1, *objectDescriptors1;
        IplImage* object=cvLoadImage(buf,CV_LOAD_IMAGE_GRAYSCALE);
        IplImage* object_color = cvCreateImage(cvGetSize(object), 8, 3);
        cvCvtColor( object, object_color, CV_GRAY2BGR );
        objectKeypoints1=0;objectDescriptors1=0;
        double tt = (double)cvGetTickCount();
        cvExtractSURF( object, 0, &objectKeypoints1, &objectDescriptors1,
storage, params );
        printf("%d Image Descriptors: %d\n", i+1,objectDescriptors[i]->total);
        tt = (double)cvGetTickCount() - tt;
        printf( "Extraction time = %gms\n", tt/(cvGetTickFrequency()*1000.));
        cvReleaseImage(&object);
        cvReleaseImage(&object_color);
        log.Log("Find object");
        //здесь собственно предстоит найти объект
        maxpair=0;
        for(il=0;il<500;il++)
        {

```



```

        vector<int> ptpairs;
        findPairs( objectKeypoints[i1], objectDescriptors[i1],
objectKeypoints1, objectDescriptors1, ptpairs );
        int n = ptpairs.size()/2;
        pai_r[i1]=n;
        if (i1>0 && n>pai_r[maxpair]) maxpair=i1;
    }
    sprintf(buf, "Object #%d", maxpair+1);
    log.Log(buf);
    all++;
    //if (pai_r[maxpair]<4) maxpair=-1;
    if (maxpair==i) all_p++;
}
}
sprintf(buf, "All #%d", all);
log.Log(buf);
sprintf(buf, "All D #%d", all_p);
log.Log(buf);

```

Листинг 10.6

В программе используются функции из предыдущего параграфа. Картинок различного типа на самом деле тестовых 1000, но у меня после 500-ой при обучении висло, поэтому без рассуждений было принято решение сокращения в два раза изображений. При обучении формировалось 500 эталонных образов с особенностями. При тестировании распознавалось $500 \times 11 = 5500$ изображений. Т.к. сравнивалось с каждым эталоном, то общее время распознавания составило 2.7 часа в одном потоке. Не очень быстро. Результаты тестирования таковы: 3775 изображений из 5500 было распознано правильно. Это 0.69% достоверности. Есть достаточно много причин такой низкой достоверности.

10.2.3. Стереозрение с использованием особенностей

В предыдущих параграфах показано, как, находя особенности объектов, можно их распознавать. Естественно возникает мысль, а что если выделить на обоих изображениях особенности и сравнить их между собой. По расхождению в пикселях найти расстояния до этих особенностей, которые и использовать, к примеру, для ориентации робота в пространстве.

Проверка гипотезы осуществлялась при помощи 3d модели. В результате было найдено 104 пары особенностей (Рис. 10.9 и 10.10)

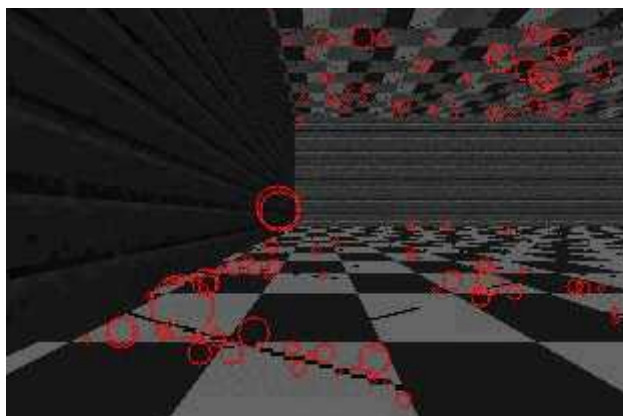


Рис. 10.9. Особенности левого изображения

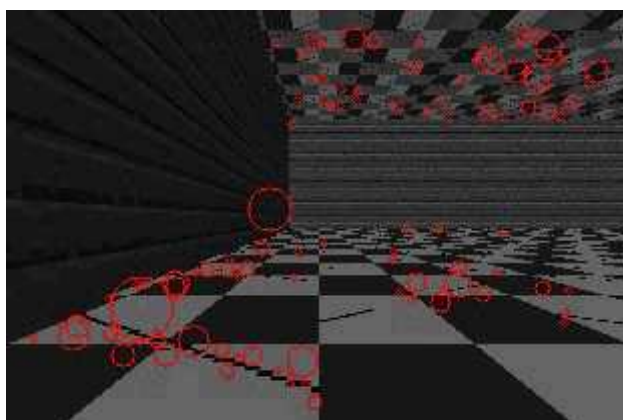


Рис. 10.10. Особенности правого изображения

Соответственно для 104 особенностей нашлось столько же трёхмерных точек, по которым была построена модель (Рис. 10.11).



Рис. 10.11. Трёхмерная модель особенностей

10.3. Слежение за баскетбольным мячом

Известен подход выделения объектов на изображениях по распределению цветовой гаммы. Данный подход является в своей основе достаточно простым, но, не смотря на это, может использоваться в ряде случаев. Иногда этот подход применяется как первоначальный этап обработки изображения, для решения более сложных задач, например детектирования лиц [1]. Такой подход может использоваться и в системах трекинга, если цвет объекта отличается от фона.

Рассмотрим задачу отслеживания движений баскетбольного мяча. Есть входной поток изображения. На каждом кадре изображения необходимо определить наличие или отсутствие баскетбольного мяча. Мяч при детектировании выделить окружностью.

Решение задачи разбивается на несколько этапов:

- выделения пикселей, соответствующих баскетбольному мячу;
- выделение контурами найденные объекты;
- нахождении контура мяча;
- построение окружности, в которую попадают все точки контура мяча.

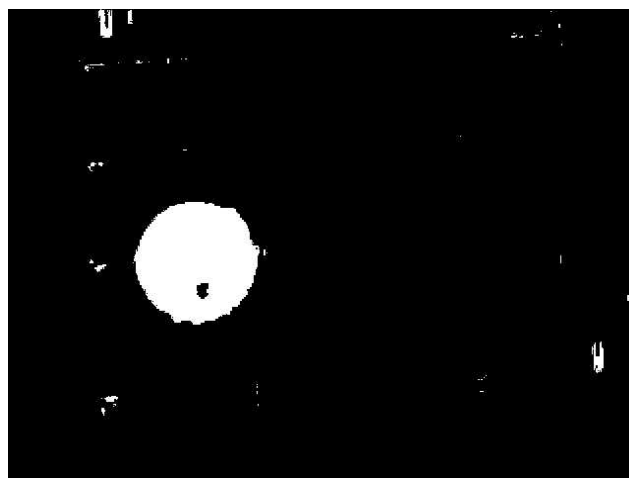
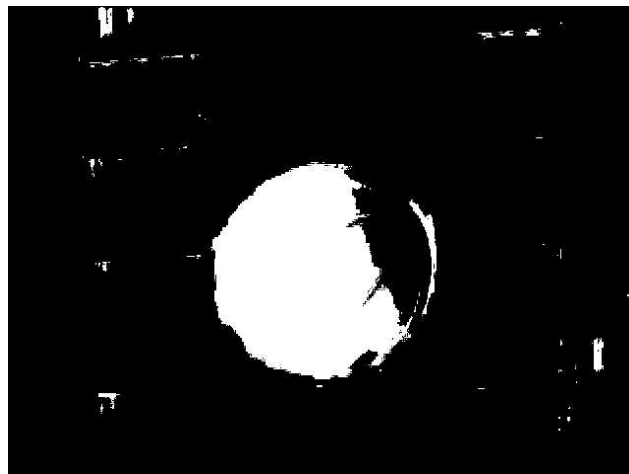
С помощью средств OpenCV данная задача решается очень просто. На рисунках 10.12а представлены примеры кадров с баскетбольным мячом. Мячу соответствуют в основном оранжевые пиксели. Поэтому для выделения большинства из них достаточно следующего условия:

$$R > 1.5 * G, R > 2 * B,$$

где R , G , и B – соответствующие компоненты пикселя.

Условие подобрано примерно и работает удовлетворительно. На рисунках 10.12б представлены результаты выделения пикселей, соответствующих мячу, что показано в виде белых пикселей на чёрном фоне.





а

б

Рис. 10.12. Исходные изображения (а), изображения с выделенными пикселями, соответствующими цвету мяча (б)

Из найденных контуров необходимо выделения максимального, который в данном примере будет соответствовать мячу (см. листинг). Результаты выделения максимального контура представлены на рисунке 10.13а.

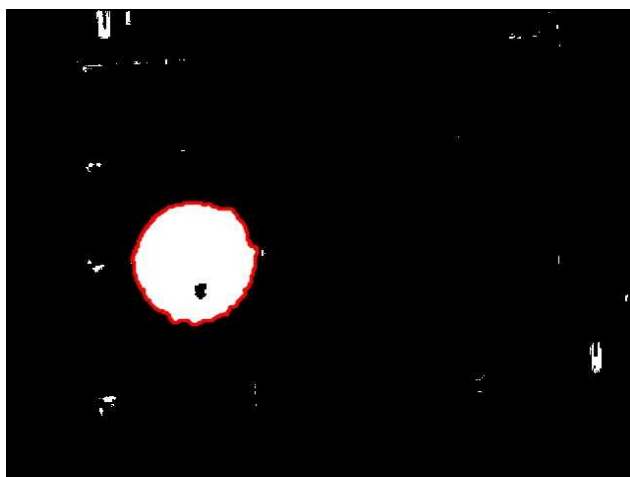
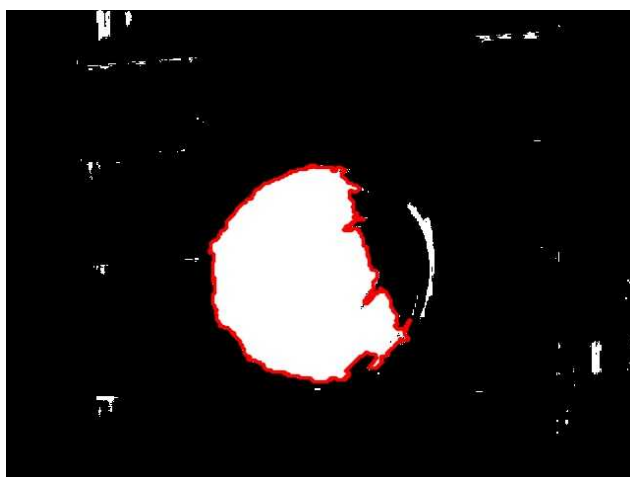
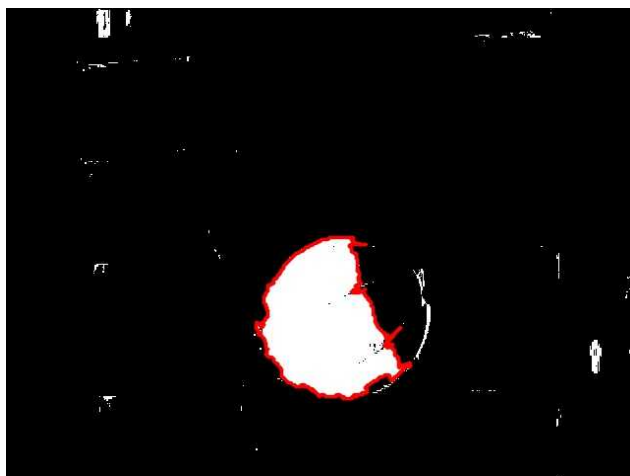
Далее необходимо найти центр и радиус окружности, соответствующей области мяча. Для этого в OpenCV есть функция `cvMinEnclosingCircle`. Ниже приведено её описание.

```
int cvMinEnclosingCircle(
    const CvArr* points,
    CvPoint2D32f* center,
    float* radius
);
```

Параметры:

points – последовательность или массив 2D точек;

center – выходной параметр: центр окружности;
radius – выходной параметр: радиус окружности.



а

б

Рис. 10.13. Выделение контура мяча (а), выделение окружности (б)

Функция ищет минимальную окружность, включающую в себя все точки последовательности. При невозможности найти окружность функция возвращает 0.

На рисунке 10.136 приведены примеры выделения окружностей для приведенных случаев.

Ниже приведен листинг программы осуществляющей слежение за баскетбольным мячом.

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#include <ctype.h>

void FindBall(IplImage* Img);
void Counter(IplImage* img);

CvPoint2D32f center;
float radius;

long pointer=0;

int main( int argc, char** argv )
{
    CvCapture* capture = 0;

    capture = cvCaptureFromAVI( "capture-1.avi" );

    cvNamedWindow( "Demo", 1 );

    for(;;)
    {
        IplImage* frame = 0;
        int i, k, c;

        frame = cvQueryFrame( capture );
        if( !frame )
            break;
    }
```



```

        FindBall(frame);
        cvShowImage( "Demo", frame );

        c = cvWaitKey(50);
        if( (char)c == 27 )
            break;
        pointer++;
    }

    cvWaitKey(0);
    cvReleaseCapture( &capture );
    cvDestroyWindow("Demo");

    return 0;
}

void FindBall(IplImage* Img)
{
    IplImage* Image=cvCreateImage( cvGetSize(Img), 8, 3 );

    cvCopy(Img, Image);

    //Теперь необходимо получить доступ к всем пикселям.
    uchar* ptr1;
    ptr1 = (uchar*) (Image->imageData );
    int i,j;
    for(i=0;i<Img->height;i++)
        for(j=0;j<Img->width;j++)
        {

            //R > 1.5*G, R > 2*B
            if (ptr1[j*3+2+i*Image->widthStep]>1.5*ptr1[j*3+1+i*Image->widthStep]
&&
                ptr1[j*3+2+i*Image->widthStep]>2*ptr1[j*3+i*Image->widthStep])
            {
                ptr1[j*3+i*Image->widthStep]=255;
            }
        }
    }

```



```

        ptr1[j*3+1+i*Image->widthStep]=255;
        ptr1[j*3+2+i*Image->widthStep]=255;
    }
    else
    {
        ptr1[j*3+i*Image->widthStep]=0;
        ptr1[j*3+1+i*Image->widthStep]=0;
        ptr1[j*3+2+i*Image->widthStep]=0;
    }
}

//Выделение контуров и поиск наибольшего контура
Counter(Image);

if (center.x>-1)
{
    CvPoint p;
    p.x=center.x;
    p.y=center.y;
    cvCircle( Img, p, radius, CV_RGB(255,0,0), 3, 8, 0 );
}
cvReleaseImage( &Image );
}

void Counter(IplImage* img)
{
    IplImage* img_gray= cvCreateImage( cvSize(img->width,img->height), 8, 1);
    CvSeq* contours = 0;

    CvMemStorage* storage = cvCreateMemStorage(0);

    cvCvtColor( img, img_gray, CV_BGR2GRAY );

    cvFindContours( img_gray, storage, &contours, sizeof(CvContour),
        CV_RETR_LIST , CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0) );
}

```



```

CvSeq* h_next=0;

//Поиск максимального контура
for( CvSeq* c=contours; c!=NULL; c=c->h_next )
{
    if (c!=contours)
    {
        //Проверяем какой контур больше
        if (h_next->total>=c->total)
        {
            h_next->h_next=h_next->h_next->h_next;
            continue;
        }
    }
    h_next=c;
}

center.x=-1;
if (h_next->total<200) return; //нет мяча - слишком маленькие контуры

cvDrawContours( img, h_next, CV_RGB(255,0,0), CV_RGB(0,255,0),2, 2, CV_AA,
cvPoint(0,0) );

//Минимальная окружность
cvMinEnclosingCircle(h_next,&center,&radius);

cvReleaseMemStorage( &storage);
cvReleaseImage( &img_gray );
}

```

Листинг 10.7

Литература:

1. Yunlong Zhao, Tat-Seng Chua. Automatic Tracking of Face Sequences in MPEG Video. In Proceedings of Computer Graphics International'2003. pp.170~175



10.4. Создание программы детектирования движения на нескольких камерах с использованием OpenCV

Сформулируем требования к программе:

- поддержка нескольких камер (например, 1-4);
- детектирование движений на всех камерах;
- сохранение отдельных кадров с движением в файлы jpg;
- сохранение отдельных кадров с движением в видео файлы;
- сохранение информации о времени фиксирования кадров в log-файлы;
- управление загрузкой процессора (было бы неправильно постоянно обрабатывать видео поток, на котором нет движений).

Программа будет писаться для Windows, но программист Linux может легко переделать её под себя, поменяв функции создания потоков и вызовы Win32 API функций системного времени и тиков процессора. Чтобы не тратить время на графический интерфейс, ограничимся файлом конфигураций и простейшей консолью. Поскольку вопросы описания работы с камерой в OpenCV достаточно широко освещены, не будем на них останавливаться, а сразу перейдём к созданию потоков для нескольких камер. Исходный код, вызываемый после захвата камер, представлен в листинге 10.8.

```
DWORD lpT;
HANDLE h;
for(i=0;i<all_camera;i++)
{
    if (!capture[i]) continue;
    NumMultiThread=i;
    h=CreateThread(NULL,0,MultiThread,NULL,0,&lpT);
    Sleep(100);
    CloseHandle(h);
}
```

Листинг 10.8

Здесь глобальной переменной **NumMultiThread** присваивается номер потока, чтобы затем в функции **MultiThread** можно было отличить, какую камеру обрабатываем. Применяя **Sleep**, засыпаем, чтобы возникло никаких нюансов вроде состояния состязания. Делаем заготовку функции **MultiThread** (Листинг 10.9).



```

DWORD WINAPI MultiThread(LPVOID)
{
    int localNumMultiThread=NumMultiThread;

    //Work with camera
    if (capture[localNumMultiThread])
    {
        IplImage* motion = 0;
        IplImage* image1 = 0;
        int detect=0;
        int waitkey=MINMSEC;

        for(;;){
            if (!capture[localNumMultiThread]) break;
            IplImage* image = cvQueryFrame( capture[localNumMultiThread]);
            if( !image ) break;
            if( !motion )
            {
                motion = cvCreateImage( cvSize(image->width,image->height), 8, 3 );
                cvZero( motion );
                motion->origin = image->origin;
            }

            detect1=update_mhi( image, motion, 30,localNumMultiThread );
            cvWaitKey(waitkey);

            if (UnloadCapture[localNumMultiThread]==0) break;
        }
        cvReleaseCapture( &capture[localNumMultiThread] );
    }

    UnloadCapture[localNumMultiThread]=-1;
    return 0;
}

```

Листинг 10.9



Рассмотрим, какие здесь появились изменения по сравнению с обычным процессом детектирования. Для отличия видеокамер используется переменная **localNumMultiThread**, **detect** служит как характеристика количества движений в кадре, **waitkey** – определяет время ожидания в миллисекундах, поскольку у нас есть задача управления загруженностью, **UnloadCapture** служит для анализа команд пользователя – когда нам надо прекратить работу. Помимо этого нам необходимо преобразовать функцию **update_mhi**, добавив параметр, определяющий поток с видеокамеры, и добавив возвращаемое значение. Для обработки различных потоков приходится хранить в памяти массив параметров (Листинг 10.10).

```
// ring image buffer
IplImage **buf[MAX_CAMERA] ;
int last[MAX_CAMERA];

// temporary images
IplImage *mhi[MAX_CAMERA]; // MHI
IplImage *orient[MAX_CAMERA]; // orientation
IplImage *mask[MAX_CAMERA]; // valid orientation mask
IplImage *segmask[MAX_CAMERA]; // motion segmentation map
CvMemStorage* storage[MAX_CAMERA]; // temporary storage
```

Листинг 10.10

Видоизменив функцию:

```
int update_mhi( IplImage* img, IplImage* dst, int diff_threshold ,int camera)
```

внутри необходимо в соответствующие места добавить [camera], при использовании этих параметров. При обработке цикла областей движений (iterate through the motion components) необходимо учитывать количество областей движений больше некоторого минимального размера.

Однако, вернёмся в моменту, когда закончился цикл создания процедур обработки видео потоков, т.е., что происходит после цикла создания потоков. Поскольку нет графического интерфейса, то можно сделать простейшую консоль (Листинг 10.11).

```
char str[21];
int k=0;
Sleep(1000);
do{
    printf("MMDT#");
    scanf("%20s", str, 21);
    k=0;
```




```

if (strcmp(str,"exit")==0) break;
if (strcmp(str,"winon")==0) {
    for(i=0;i<allcam;i++)
        WindowCapture[i]=1;
    k=1;
}
if (strcmp(str,"winoff")==0) {
    for(i=0;i<allcam;i++)
        WindowCapture[i]=0;
    k=1;
}
if (k==0) printf("Unknown command\n");
}while(1);

```

Листинг 10.11

При команде **exit** необходимо завершить все потоки и выйти из программы (Листинг 10.12). При команде **winon** происходит отображение графических окон с web-камер, а при **winoff** – закрытие.

```

if (all_camera>0)
for (i=0;i<all_camera;i++)
    if (capture[i])
    {
        UnloadCapture[i]=0;
        do{
            Sleep(100);
        }while(UnloadCapture[i]!=-1);
    }
return 0;

```

Листинг 10.12

Процесс управления нагрузкой центрального процессора в данном случае является простейшей задачей управления процессом распознавания образов . Когда в кадре не будет движений, то задержка кадров будет увеличена до максимума (**MAXMSEC**), в противном случае уменьшена до минимума (**MINMSEC**). Реализовать это достаточно просто – после получения количества областей движения необходимо вставить следующий код (Листинг 10.13).

```

if (detect>0) detect--;
if (detect1>0) detect=5;
if (detect>0) waitkey=MINMSEC;
else waitkey=MAXMSEC;

```

Листинг 10.13

Здесь **detect1** показывает количество движений в области, а **detect** определяет порог, после скольких кадров без движений происходит увеличение задержки.

Для того, чтобы была возможность понаблюдать за видео потоками с камер, реализуем возможность включения и выключения окон (команды **winoff** и **winon**), что показано ниже (листинг 10.14).

```

//Вызов перед циклом обработки кадров
char buf[256];
sprintf(buf, "Camera #%d", localNumMultiThread);
if (localWindowCapture) cvNamedWindow( buf, 1 );
...
//Вызов в цикле обработки кадров
if (localWindowCapture!=WindowCapture[localNumMultiThread])
{
    if (localWindowCapture) cvDestroyWindow(buf);
    else cvNamedWindow( buf, 1 );
    localWindowCapture=WindowCapture[localNumMultiThread];
}
if (localWindowCapture) cvShowImage( buf, image);
...
//Вызов после цикла обработки кадров
if (localWindowCapture) cvDestroyWindow(buf);

```

Листинг 10.14

Для записи кадров в файлы jpg в программе будут учтены настройки пути до хранения и включенность режима записи. В листинге 10.15 приведён код, вызываемый внутри цикла обработки кадров для сохранения в jpg файлы.

```

if (detect>0 && OUTJPG)
{

```



```

strcpy(buf1, PATHJPG);
sprintf(buf2, "cam%dn", localNumMultiThread);
strcat(buf1, buf2);
FILE *f;
strcpy(buf3, buf1);
for(long ii=JPG_CAM+1; ii<MAX_JPG_CAM; ii++)
{
    strcpy(buf1, buf3);
    sprintf(buf2, "%d.jpg", ii);
    strcat(buf1, buf2);
    f=fopen(buf1, "r");
    if (f==NULL) break;
    fclose(f);
}
cvSaveImage(buf1, image1);
}

```

Листинг 10.15

Условие срабатывает только, когда задержка кадров минимальна и включен режим вывод jpg файлов **OUTJPG**. Затем создаётся имя файла по заданному в настройках пути **PATHJPG**, которое в цикле сравнивается с существующими на диске файлами, и, если такие есть, то генерируется следующее имя. После чего изображение сохраняется в файл.

Для того, чтобы записывать кадры в видеофайл первоначально необходимо осуществить захват устройства (Листинг 10.16).

```

//Первоначально определяется перед циклом
CvVideoWriter* cvVideoWriter
...
//Внутри цикла
if (OUTAVI)
{
    if (!cvVideoWriter)
    {
        strcpy(buf1, PATHAVI);
        sprintf(buf2, "cam%d.avi", localNumMultiThread);
        strcat(buf1, buf2);
    }
}

```



```

        cvVideoWriter=cvCreateVideoWriter(buf1, CV_FOURCC(CODEC[0], CODEC[1],
CODEC[2], CODEC[3]) ,10,
        cvSize(image ->width/2,image->height/2));
        logpolar_frame = cvCreateImage(cvSize(image ->width/2,image-
>height/2),IPL_DEPTH_8U,3);
        cvZero(logpolar_frame );
        logpolar_frame->origin = logpolar_frame->origin;
        cvLogPolar( imagel, logpolar_frame,cvPoint2D32f(imagel->width/2,
        imagel->height/2),40,
        CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS );
    }
}

```

Листинг 10.16

Первоначально создаётся имя и путь до видеофайла, а затем происходит захват устройства (**cvCreateVideoWriter**), причём наименование кодека будет браться из файла и храниться в массиве **CODEC**. Для уменьшения размеров видеофайла уменьшим в 2 раза все размеры картинки. А затем идёт некоторая странная вещь, которую я подсмотрел в книге «Learning OpenCV». Это вызов функции **cvLogPolar**. Сама по себе эта функция нам не нужна, однако она видимо меняет какие-то настройки **IpplImage**, какие я не разобрался – посмотрел в Debug, но все параметры вроде бы не меняются, может быть идёт какое-то выравнивание на границу памяти, но я не стал смотреть. Но без этой функции дальнейшие попытки записи в видеофайл приводят к неудаче. Сама запись довольно проста (листинг 10.17).

```

if (detect>0) {
    if (OUTAVI) {
        cvResize(imagel, logpolar_frame, 2);
        cvWriteFrame(cvVideoWriter,logpolar_frame);
    }
}

```

Листинг 10.17

Текст результирующей программы детектирования движений вы можете скачать здесь:

Multiple Motion Detect <http://vidikon.com/download/multimd.zip>

На рисунке 10.14 представлена система для тестирования программы с 4 камерами (1 встроенная).



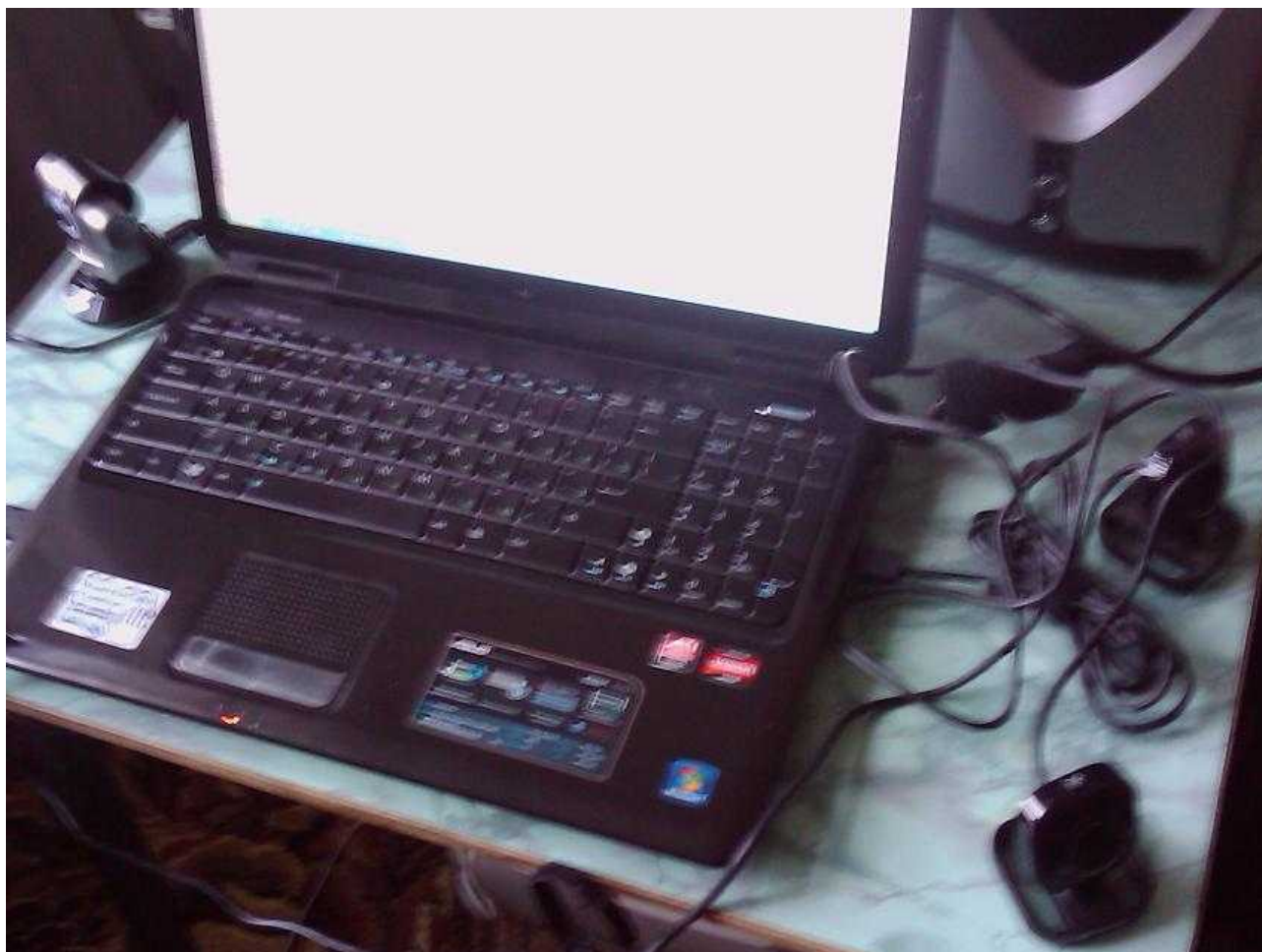
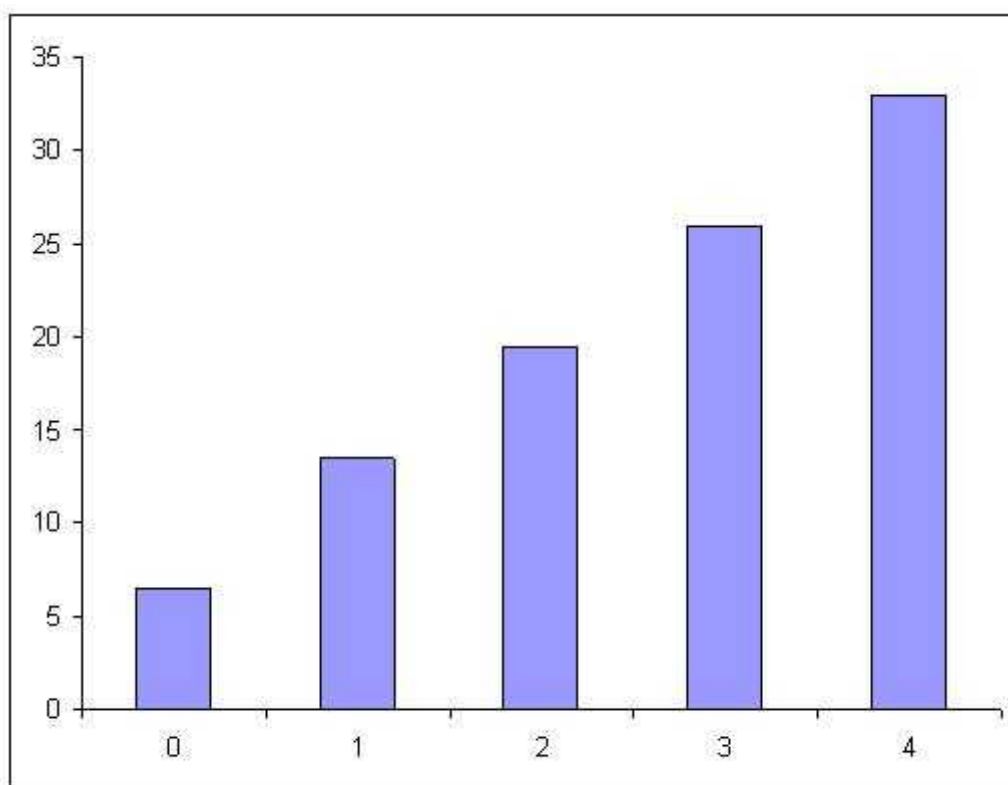


Рис. 10.14. Система для тестирования

Программа реализует самый простейший способ управления процессом распознавания образов в реальном времени. Когда на камерах нет движения, то частота кадров снижается до минимальной (т.е. максимальная задержка). В противном случае увеличивается. Результаты смотрите на рисунке 10.15.



Количество камер, на которых обнаружено движение	Процент загрузки CPU
0	4-8%
1	10-16%
2	17-21%
3	22-29%
4	31-34%

Рис. 10.15. Загруженность CPU

Понятно, что если не управлять детектированием движения загрузка будет постоянно максимальной, что для описываемого случая приемлемо, но в ситуации, когда максимальная загрузка колеблется около 100% - это необходимое решение.

10.5. Подсчёт объектов (людей), пересекающих линию (People counting)

Самый простейший путь при подсчёте объектов – это использование детектирования движений. Например, используя ту же функцию **update_mhi** из примеров OpenCV. Собственно решение напрашивается само – следить за выделяемыми областями движения, и как только происходит

пересечение заданной линии, то увеличивать значение счётчика. Однако функция **update_mhi** не предназначена для слежения за объектами, поэтому эту часть придётся делать самостоятельно. На рисунке 10.16 показано обнаруженные центры движений (а), и изменения положений центров движений (б).

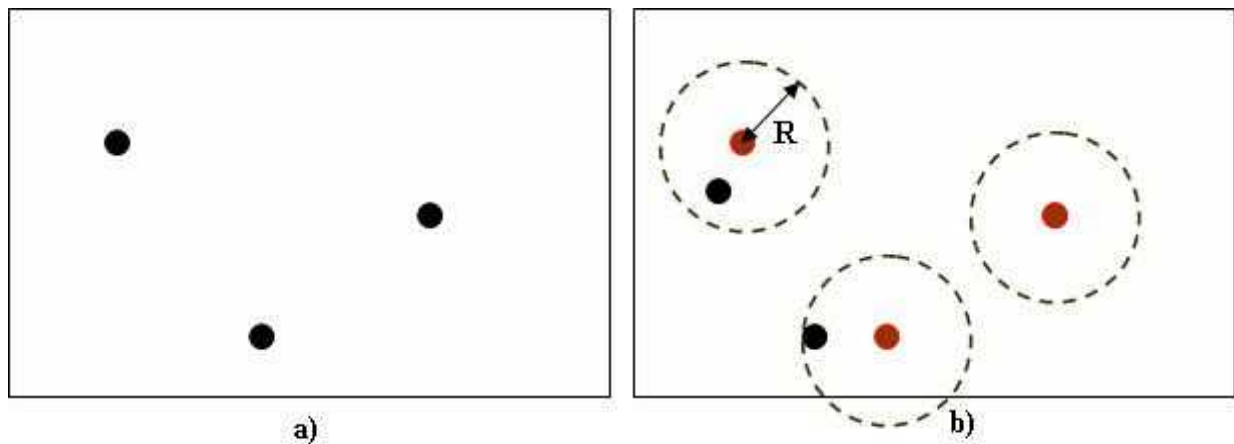


Рис. 10.16. Центры движений в кадре (а), изменение движений (б)

Красные центры – старое местоположение, чёрные – новое. Одной из красных точек не соответствует чёрной, т.к. объект остановился и не двигается. **R** – максимальное расстояние, на которое может сместиться объект за один кадр.

Для наблюдения за объектами будет использоваться следующая структура:

```
struct _OBJECT_
{
    int x,y; //Координаты центра объекта
    int w,h; //Ширина, высота объекта
    int timer; //Задержка, позволяющая следить за объектом после остановки
    int num; //Номер объекта
    int object; //Соответствие с новым объектом
    int inn; //Переменная, показывающая где находится объект относительно линии
    пересечения
} VObject[MAX_OBJECTS], BVObject[MAX_OBJECTS];
```

VObject – текущий массив объектов, **BVObject** – предыдущий массив объектов. При каждом анализе детектируемых зон, необходимо сначала сохранить предыдущий массив, например так:

```
memcpy(BVObject,VObject,sizeof(_OBJECT_)*MAX_OBJECTS);
```

Если в предыдущем массиве нет ни одного объекта, то при анализе детектируемых зон можно вызвать следующий листинг:

```

if (all_object<MAX_OBJECTS) {
    VObject[all_object].num=end_object;
    VObject[all_object].timer=timer_1;
    VObject[all_object].x=xx;
    VObject[all_object].y=yy;
    VObject[all_object].w=comp_rect.width;
    VObject[all_object].h=comp_rect.height;
    VObject[all_object].object=-1;
    VObject[all_object].inn=2;
    cur_object=all_object;
    all_object++;
    end_object++;
}

```

Листинг 10.18

Здесь, **end_object** – глобальный счётчик номеров объектов (всегда увеличивается); **timer_1** – максимальное время задержки, после которого объект, которому не соответствия движения, уничтожается; **xx, yy** – текущие координаты центра движения; **comp_rect** – размер зоны движения.

Если от предыдущего кадра сохранились объекты, то необходимо проверять соответствие со старыми объектами:

```

min=-1;
for (j=0; j<local_all_object; j++)
    if (BVObject[j].object==-1)
    {
        dd=LengthLine(xx,yy,BVObject[j].x,BVObject[j].y);
        if (dd<maxdist && (min==-1 || dd1>dd))
        {
            min=j;
            dd1=dd;
        }
    }

if (min==-1) {
    VObject[all_object].num=end_object;
}

```




```

    VObject[all_object].inn=2;
    end_object++;
}
else {
    VObject[all_object].num=BVObject[min].num;
    VObject[all_object].inn=BVObject[min].inn;
    BVObject[min].object=0;
}
VObject[all_object].timer=timer_1;
VObject[all_object].x=xx;
VObject[all_object].y=yy;
VObject[all_object].w=comp_rect.width;
VObject[all_object].h=comp_rect.height;
VObject[all_object].object=-1;
cur_object=all_object;
all_object++;

```

Листинг 10.19

Здесь, **local_all_object** – общее количество старых объектов. Если указатель ссылки на новый объект (**BVObject[j].object**) равен -1, то сравнивается его удалённость с текущим объектом, если она минимальна и меньше расстояния **R**, заданного переменной **maxdist**, то соответствие найдено. Если соответствие не было найдено, то создаётся новый объект, иначе текущему объекту присваиваются номера и положения относительно граничной линии старого объекта.

Определение пересечения линии достаточно просто – необходимо сравнивать значения **inn** текущего и старого объекта. Определить **inn** относительно положения текущей линии можно по следующему листингу:

```

float ang=MakePolarF(center.x-Center.x,center.y-Center.y);
byte inn=0;
if ((ang>In_ && ang<In_+PI) || (In_>PI && ang<PI-In_)) {
    inn=1;
}
if (VObject[cur_object].inn==0 && inn==1) All_In++;
if (VObject[cur_object].inn==1 && inn==0) {
    All_Out++;
}

```



```
VObject[cur_object].inn=inn;
```

Листинг 10.20

Здесь, **center** – это центр области движения; **Center** – центр заданной нами линии; **In_** - угол в радианах вектора из центра линии в одну из сторон линии.

Этот метод прост и не лишён недостатков, возможно он не всегда работает

10.6. Детектирование QR Code с помощью средств OpenCV

Распознавание двумерных кодов является актуальной задачей, впрочем, уже решенной на ряде устройств и персональных компьютерах. Однако на настоящий момент отсутствуют BSD версии библиотек распознавания, а ограниченные лицензии в большинстве случаев не позволяют использовать открытые исходные кода для своих целей.

Популярный в Японии Qr Code является одним из наиболее оптимальных кодов, хотя при малых размерах кода квадраты-мишени отнимают много места. На рисунке 10.17 приведен пример кода Qr с закодированной фразой «Test code».



Рис. 10.17. Пример кода Qr («Test code»)

Описание принципов кодирования-декодирования кода доступно в спецификации [1]. Принципы нахождения границ кода с произвольным углом наклона и поворота камеры описаны в работах [2, <http://blog.vidikon.com/?p=411>]. Однако возникает вопрос: можно ли с помощью средств известной библиотеки OpenCV упростить распознавание кода? Можно. И пример такого упрощения представлен в библиотеке libdecodeqr.

Для начала упростим задачу. Пусть необходимо найти местонахождение 4-х граничных точек кода, причем ось камеры примерно перпендикулярна плоскости кода, а сам код может быть повернут в плоскости не более, чем на 30 градусов (по сравнению с рис.1 в обе стороны). Процесс нахождения местоположения кода можно разбить на несколько этапов.

1. *Перевод в монохромное изображение.* Осуществляется перевод изображения из полноцветного в градации серого с помощью функции `cvCvtColor`, а затем перевод в монохромное с помощью функций `cvThreshold` или `cvAdaptiveThreshold`.

2. *Выделение контуров.* На преобразованном изображении находятся контура с использованием функции `cvFindContours` (....).

3. *Поиск квадратов.* Далее необходимо найти, какие контура можно условно считать квадратами. Для этого необходимо выяснить ограничивающую зону контура с помощью `cvBoundingRect`, воспользоваться функцией `cvContourArea` для вычисления внутренней области контура.

```
CvRect cvBoundingRect(  
    CvArr* points,  
    int update=0  
);
```

Функция возвращает ограничивающий четырехугольник для массива точек.

```
double cvContourArea(  
    const CvArr* contour,  
    CvSlice slice=CV_WHOLE_SEQ  
);
```

Функция возвращает площадь контура `contour`.

После, если площадь контура не намного отличается от ограничивающего четырёхугольника, то можно считать его квадратом.

4. *Проверка на наличие внутренних квадратов.* Если у найденных квадратов есть внутренние квадраты, то будем считать эти квадраты мишенями. Найденных мишеней в нашем случае должно быть 3.

5. *Поиск четырех точек.* Для поиска 4-х точек необходимо определить 4 угловые точки для каждой мишени с помощью функций `cvMinAreaRect2` и `cvBoxPoints`.

```
CvBox2D cvMinAreaRect2(  
    const CvArr* points,  
    CvMemStorage* storage=NULL  
);
```



Функция возвращает прямоугольник с минимальной площадью, ограничивающей точки points.

```
void cvBoxPoints(  
    CvBox2D box,  
    CvPoint2D32f pt[4]  
);
```

Функция для найденного прямоугольника CvBox2D возвращает 4 угловые точки pt.

Из найденных точек для 3-х мишеней нехитрыми действиями выбираются 3 крайние точки кода. 4-ая точка может быть получена двумя способами: а) воспользоваться функцией cvBoxPoints для ограничения всех точек всех мишеней – в результате будет найдена 4 точка (но не очень точно); б) по двум крайним точкам нижней и правой мишеней определить направляющую линию, на линии пересечения которой будет находиться искомая точка. результат смотрите на рисунке 10.18.

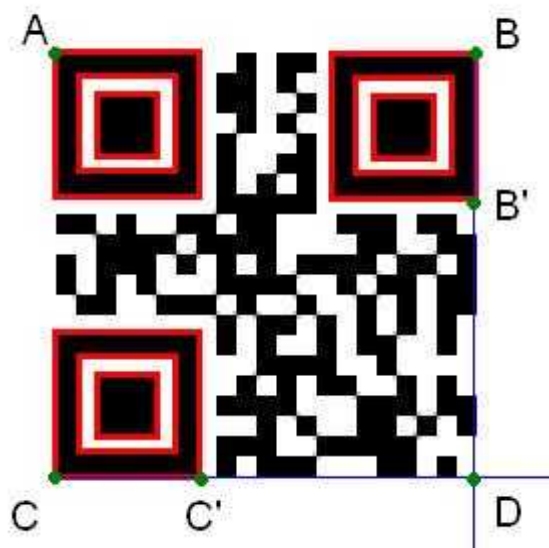


Рис. 10.18. Найденные крайние точки кода

При несильном искажении кода по полученным точкам можно разбить код на сетку, в которой определить, пустая или полная ячейка. При сильных искажениях, сетку придется масштабировать с разными размерами ячеек, синхронизируясь по дополнительным элементам.

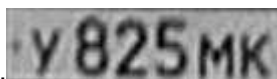
Литература:

1. ISO/IEC 18004:2006 Information technology — Automatic identification and data capture techniques — QR Code 2005 bar code symbology specification.
2. «Barcode readers using the camera device in mobile phones» Ohbuchi, E. Hanaizumi, H. Hock, L.A. ,Cyberworlds, 2004 International Conference on, page 260 – 265.
3. <http://blog.vidikon.com/?p=411> Распознавание двумерных штрих кодов с произвольным углом наклона и поворота камеры

10.7. Распознавание текста с использованием шаблонов

Конечно, известны методы распознавание текста на базе нейронных сетей, но существует ещё более простой метод (в плане обучения), основанный на сравнении с эталоном (шаблоном). В ряде случаев сравнение с шаблоном будет достаточно для распознавания текста, например, при распознавании автомобильных номеров.

Предположим на некотором изображении есть текст:



И нам естественно необходимо его распознать. Но первоначально необходимо найти местонахождение букв и цифр. Для этого изображение надо предварительно обработать, а именно – привести изображение к монохромному. В библиотеке OpenCV есть для этого несколько функций: `cvThreshold()`, `cvAdaptiveThreshold()`. Данные функции не раз использовались выше, поэтому не будем останавливаться на них подробнее. Здесь однако стоит отметить, что при работе с изображениями данного типа (с текстом) стоит использовать не адаптивную функцию, а обычную `cvThreshold()`, впрочем выбирая оптимальный порог, например, используя метод Отсу.

Далее можно выделить контуры с использованием функции `cvFindContours()`. В большинстве случаев будет много всякого шума, но его частично можно удалить, зная приблизительные размеры букв.



После удаления ненужных контуров похожим образом останутся нужные контура находимых букв. Используя функцию `cvBoundingRect()`, получаем области нахождения символов.



После этого можно приступить к анализу найденных областей символов, но для этого необходимо составить шаблоны образов. Шаблоны составляются в виде массива, например, как представлено ниже:

```
-1 , 1 , 1 , 1 , -1  
1 , -1 , -1 , -1 , 1  
1 , -1 , -2 , -1 , -1  
1 , -1 , -2 , -2 , -2  
1 , -1 , -2 , -1 , -1  
1 , -1 , -1 , -1 , 1  
-1 , 1 , 1 , 1 , -1
```

Как вы уже догадались, это шаблон буквы «С». Для того, чтобы сравнить область изображения с символом необходимо привести эту область к размеру шаблона или наоборот – шаблон к размеру

области. Можно просто использовать известные методы увеличения, или написать самому. Ниже представлен листинг, в котором осуществляется приведение к размеру шаблона (в данном случае 5 на 7).

```
double a[5*7];
double a1[4];
uchar* ptr = (uchar*) (Image->imageData);
int x,y;
int x1,y1;
double x1_,y1_;
int x2,y2;
double x2_,y2_;
...
memset(a,0,5*7*sizeof(double));
for(y=0;y<Rect.height;y++){
    for(x=0;x<Rect.width;x++){
        if (ptr[x+Rect.x+(y+Rect.y)* Image->widthStep]==0) continue;
        memset(a1,0,4*sizeof(double));
        x1_=(double)x*ratioW;
        y1_=(double)y*ratioH;
        x1=int(x1_);y1=int(y1_);
        x2_=(double)(x+1)*ratioW;
        y2_=(double)(y+1)*ratioH;
        x2=int(x2_);y2=int(y2_);
        if (x2==x1) { a1[0]=1;a1[1]=0;}
        else {
            a1[0]=(double)(x1+1-x1_)/(x2_-x1_);
            a1[1]=(double)(x2_-x2)/(x2_-x1_);
        }
        if (y2!=y1){
            a1[2]=(double)a1[0]*(y2_-y2)/(y2_-y1_);
            a1[3]=(double)a1[1]*(y2_-y2)/(y2_-y1_);
            a1[0]=(double)a1[0]*(y1+1-y1_)/(y2_-y1_);
            a1[1]=(double)a1[1]*(y1+1-y1_)/(y2_-y1_);
        }
        a[x1+y1*7]+=a1[0];
    }
}
```



```
if (x1<4)
    a[x1+1+y1*7]+=a1[1];
if (y1<6)
    a[x1+(y1+1)*7]+=a1[2];
if (x1<4 && y1<6)
    a[x1+1+(y1+1)*7]+=a1[3];
}
```

Листинг 10.21

В листинге перебираются каждые точки, относящиеся к символу, после чего происходит их отнесение к той или иной ячейки изображения, причем не всегда полностью, но и частично – часть пикселя в одну ячейку, часть в другую. Далее можно сравнить полученный результат с каждым из эталонов, просто перемножив соответствующие элементы матрицы. И по наибольшему совпадению выбрать эталон.



СОДЕРЖАНИЕ

1. Введение	3
1.1. Подключение OpenCV к программе	3
1.2. OpenCV 2.0: проблемы при установке в Windows	3
1.3. Статическое подключение OpenCV 1.1 к проекту в VC 2003	4
2. HighGUI	6
2.1. Простой GUI	6
2.2. Загрузка и сохранение изображений	8
2.3. HighGUI: Видео ввод/вывод	10
3. Обработка и трансформация изображений	19
3.1. Бинаризация изображений	19
3.2. Расширенные морфологические преобразования	26
3.2.1 Пример осуществления операции морфологического градиента	28
3.3. Преобразование Фурье	30
3.4. Прозрачный цвет в OpenCV	36
4. Гистограммы яркости	37
4.1. Гистограммы яркости в OpenCV	37
4.2. Сравнение гистограмм	39
5. Контурный анализ	42
5.1. Использование контуров	42
5.1.1 Удаление мелких контуров	48
5.2. Моменты в OpenCV	49
5.2.1 Расчет моментов для символов	53
6. Отличия между кадрами и сегментация изображений	60
6.1. Выделение отличных от фона объектов	60
6.1.1 Движение курсором мышки при помощи пальца или фломастера перед камерой с неоднородным фоном	64
6.2. Отличия от фона OpenCV	69
6.3. Watershed сегментация	72
7. Трекинг и движение	76
7.1. Детектирование движений	76
7.2. Оптический поток	83
7.3. Управляем презентацией взмахом руки	86
7.4. Слежение за точками	91
8. Стереозрение	100
8.1. Стереозрение средствами OpenCV	100
8.2. Стереозрение: перевод «различий» в реальные расстояния	103
8.3. Функции стереозрения в OpenCV	106
9. Машинное обучение	112
9.1. Машинное обучение в OpenCV	112
9.2. Нейронные сети в OpenCV (Neural networks)	114
10. Разное и практические примеры	118
10.1. Детектирование лиц и глаз	118
10.1.1. Можно ли с помощью OpenCV идентифицировать человека по лицу?	124
10.1.2. Реализация трекинга лица	127
10.2. Особенности и Функция ExtractSURF	130
10.2.1. Нахождение объектов на изображении с использованием особенностей	132
10.2.2. Использование особенностей для распознавания образов	142
10.2.3. Стереозрение с использованием особенностей	144
10.3. Слежение за баскетбольным мячом	146
10.4. Создание программы детектирования движения на нескольких камерах с использованием OpenCV	153
10.5. Подсчёт объектов (людей), пересекающих линию (People counting)	161
10.6. Детектирование QR Code с помощью средств OpenCV	165
10.7. Распознавание текста с использованием шаблонов	168

