OpenCV Learning Notes

Contents

[1. Introduction 2](#_Toc464331016)

[2. Introduction to OpenCV 2.x 3](#_Toc464331017)

[(1) Read Image and Show In a Window: 3](#_Toc464331018)

[(2) Read Video and Display In a Window: 3](#_Toc464331019)

[(3) Process Image Example: 3](#_Toc464331020)

[3. Getting to Know OpenCV 5](#_Toc464331021)

[(1) Overview of Basic Type 5](#_Toc464331022)

[a. Point 5](#_Toc464331023)

[b.Scalar 5](#_Toc464331024)

[c. Size 5](#_Toc464331025)

[d. Rect 5](#_Toc464331026)

[e. RotatedRect 5](#_Toc464331027)

[f. The Fixed Matrix Classes 6](#_Toc464331028)

[g. The Fixed Vector Classes 6](#_Toc464331029)

[h. The Complex Number Classes 6](#_Toc464331030)

[i. class cv::TermCriteria 6](#_Toc464331031)

# 1. Introduction

reference book:

Learning OpenCV (Computer Vision in C++ with the OpenCV Library)

Get OpenCV source code:

git clone <https://github.com/Itseez/opencv.git>

# 2. Introduction to OpenCV 2.x

tip: OpenCV function live with namespace called cv.

## (1) Read Image and Show In a Window:

Mat img = **imread**( argv[1], -1 );

if( img.empty() ) return -1;

**namedWindow**( "Example2", WINDOW\_AUTOSIZE );

**imshow**( "Example2", img );

waitKey( 0 );

**destroyWindow**( "Example2" );

## (2) Read Video and Display In a Window:

cv::namedWindow( "Example3", cv::WINDOW\_AUTOSIZE );

cv::**VideoCapture** cap;

cap.open( string(argv[1]) );

**cv::Mat frame**;

while( 1 ) {

**cap >> frame**;

if( !frame.data ) break; // Ran out of film

cv::imshow( "Example3", frame );

if( cv::waitKey(33) >= 0 ) break;

}

get video properties:

int frame\_numbers = (int) g\_cap.get(cv::CAP\_PROP\_FRAME\_COUNT);

int frame\_width = (int) g\_cap.get(cv::CAP\_PROP\_FRAME\_WIDTH);

int frame\_height = (int) g\_cap.get(cv::CAP\_PROP\_FRAME\_HEIGHT);

## (3) Process Image Example:

Gaussian Blur One Image

cv::Mat out;

// Do the smoothing

// Could use GaussianBlur(), blur(), medianBlur() or bilateralFilter().

cv::**GaussianBlur**(frame, out, cv::Size(5, 5), 3, 3);

cv::**GaussianBlur**(out, out, cv::Size(5, 5), 3, 3);

cv::imshow("Example2\_4", out);

Gaussian Blur and Downsample:

cv::pyrDown( img, img2);

Canny Edge Detector:

cv::Canny( img\_gry, img\_cny, 10, 100, 3, true );

Access several pixel / modify pixel:

cv::Vec3b intensity = frame.at< cv::Vec3b >(y, x);

uchar blue = intensity.val[0]; // We could write img\_rgb.at< cv::Vec3b >(x,y)[0]

uchar green = intensity.val[1];

uchar red = intensity.val[2];

# 3. Getting to Know OpenCV

## (1) Overview of Basic Type

a. Point

two kinds of point: two dimensional and three dimensional

cv::Point2i

cv::Point3i

dot product and cross product:

cv::Point3i p0(1, 2, 3);

cv::Point3i p1(4, 5, 6);

float x = p0.dot(p1);

cv::Point3i cy(p0.cross(p1));

### b.Scalar

The class cv::Scalar is really a four-dimensional point class.

### c. Size

The size classes are, in practice, similar to the corresponding point classes, and can be cast to and from

them. The primary difference between the two is that the point class’ data members are named x and y,

while the corresponding data members in the size classes are named width and height

Value constructors cv::Size2f sz( w, h );

Member access sz.width; sz.height;

Compute area sz.area();

### d. Rect

The rectangle classes include the members x and y of the point class (representing the upper-left corner of the rectangle) and the members width and height of the size class (representing the extent of the rectangle). The rectangle classes, however, do not inherit from the point or size classes, and so in general they do not inherit operators from them.

Value constructors cv::Rect( x, y, w, h );

Construct from origin and size cv::Rect( p, sz );

Construct from two corners cv::Rect( p1, p2 );

Member access r.x; r.y; r.width; r.height;

Compute area r.area();

Extract upper-left corner r.tl();

Extract lower-right corner r.lr;

Determine if point p is inside of rectangle r r.contains( p );

### e. RotatedRect

The cv::RotatedRect class is one of the few classes in the C++ OpenCV interface that is not a

template underneath. It is a container, which holds a cv::Point2f called center, a cv::Size2f

called size, and one additional float called angle, with the latter representing the rotation of the

rectangle around center. One very important difference between cv::RotatedRect and cv::Rect

is the convention that a cv::RotatedRect is located in “space” relative to its center, while the

cv::Rect is located relative to its upper-left corner.

### f. The Fixed Matrix Classes

The fixed matrix classes are for matrices whose dimensions are known at compile time (hence “fixed”). As

a result, all memory for their data is allocated on the stack, which means that they allocate and clean up

quickly. Operations on them are fast and there are specially optimized implementations for small matrices

(2-by-2, 3-by-3, etc.). The

### g. The Fixed Vector Classes

The fixed vector classes are derived from the fixed matrix classes. They are really just convenience

functions for cv::Matx<>. In the proper sense of C++ inheritance, it is correct to say that the fixed vector

template cv::Vec<> is a cv::Matx<> whose number of columns is one. The readily available aliases

for specific instantiations of cv::Vec<> are of the form cv::Vec{2,3,4,6}{b,s,w,i,f,d}, where the

last character has the usual meanings (with the addition of w, which indicates an unsigned short).

### h. The Complex Number Classes

One more class type should be included in the basic types: the complex number classes. The OpenCV

complex number classes are not identical to, but are compatible with, and can be cast to and from, the

classes associated with the STL complex number class template complex<>. The most substantial

difference between the OpenCV and STL complex number classes is in member access. In the STL classes,

the real and imaginary parts are accessed through the member functions real() and imag(), while in

the OpenCV class, they are directly accessible as (public) member variables re and im.

### i. class cv::TermCriteria

Many algorithms require a stopping condition to know when to quit. Generally, stopping criteria take the

form of either some finite number of iterations that are allowed (called COUNT or MAX\_ITER) or some

kind of error parameter that basically says “if you are this close, you can quit” (called EPS—short for

“epsilon,” everyone’s favorite tiny number). In many cases, it is desirable to have both of these at once, so that if the algorithm never gets “close enough,” then it will still quit at some point.

### j. class cv::Range

The cv::Range class is used to specify a continuous sequence of integers. cv::Range objects have two

elements, start and end

### k. The cv::Ptr template, and Garbage Collection 101

One very useful object type in C++ is a “smart” pointer.7 This pointer allows us to create a reference to a

thing, and then pass that around. You can create more references to that thing, and then all of those

references will be counted. As references go out of scope, the reference count for the smart pointer is

decremented. Once all of the references (instances of the pointer) are gone, the “thing” will automatically be cleaned up (deallocated). The programmer doesn’t have to do this bookkeeping anymore.

### l. class cv::Exception and Exception Handling

OpenCV uses exceptions to handle errors. OpenCV defines its own exception type cv::Exception,

which is derived from the STL exception class std::exception. Really, this exception type has

nothing special about it, other than being in the cv:: namespace and so distinguishable from other objects that are also derived from std::exception.

### m. The cv::DataType<> Template

When OpenCV library functions need to communicate the concept of a particular data type, they do so by creating an object of type cv::DataType<>. cv::DataType<> itself is a template, and so the actual

objects passed around are specializations of this template. This is an example of what in C++ are generally called traits. This allows the cv::DataType<> object to both contain runtime information about the type, as well as to contain typedef statements in its own definition that allow it to refer to the same type at compile time.

### n. class InputArray and class OutputArray

Many OpenCV functions take arrays as arguments and return arrays as return values, but in OpenCV, thereare many kinds of arrays. We have already seen that OpenCV supports some small array types

(cv::Scalar, cv::Vec, cv::Matx), and STL’s std::vector<> in addition to the large array

types in the next section (cv::Mat and cv::SparseMat). In order to keep the interface from becoming

onerously complicated (and repetitive), OpenCV defines the types cv::InputArray andcv::OutputArray. In effect, these types mean “any of the above” with respect to the many array forms supported by the library. There is even a cv::InputOutputArray, specifying an array for in place computation.

### o. class cv::Mat: N-Dimensional Dense Arrays