Seeing With OpenCV

A Computer-Vision Library

by Robin Hewitt

PART 1

OpenCV — Intel's free, open-source computer-vision library — can greatly simplify computer-vision programming. It includes advanced capabilities — face detection, face tracking, face recognition, Kalman filtering, and a variety of artificial-intelligence (AI) methods — in ready-to-use form. In addition, it provides many basic computer-vision algorithms via its lower-level APIs.

A good understanding of how these methods work is the key to getting good results when using OpenCV. In this five-part series, I'll introduce you to OpenCV and show you how to use it to implement face detection, face tracking, and face recognition. Then, I'll take you behind the scenes to explain how each of these methods works and give you tips and tricks for getting the most out of them.

This first article introduces OpenCV. I'll tell you how to get it and give you a few pointers for setting it up on your computer. You'll learn how to read and write image files, capture video, convert between color formats, and access pixel data — all through OpenCV interfaces.

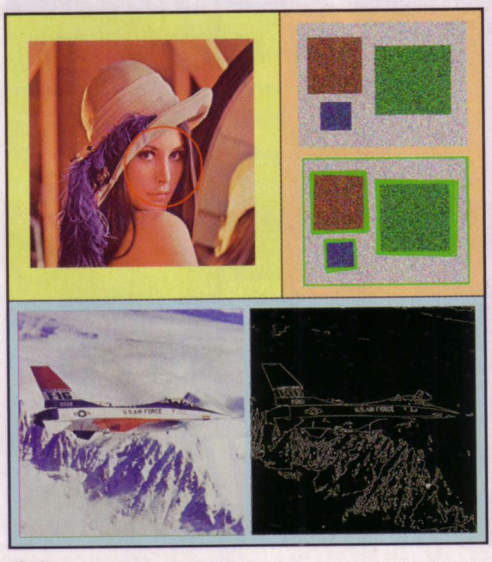

FIGURE 1. Among OpenCV's many capabilities are face detection (top left), contour detection (top right), and edge detection (bottom).

OpenCV Overview

OpenCV is a free, open-source computer vision library for C/C++ programmers. You can download it from **http://sourceforge.net/projects/opencvlibrary**.

Intel released the first version of OpenCV in 1999. Initially, it required Intel's Image Processing Library. That dependency was eventually removed, and you can now use OpenCV as a standalone library.

OpenCV is multi-platform. It supports both Windows and Linux, and more recently, MacOSX. With one exception (CVCAM, which I'll describe later in this article), its interfaces are platform independent.

Features

OpenCV has so many capabilities, it can seem overwhelming at first. Fortunately, you'll need only a few to get started. I'll walk you through a useful subset in this series.

Here's a summary of the major functionality categories in OpenCV, version 1.0, which was just released at the time of this writing:

Image and video I/O

These interfaces let you read in image data from files, or from live video feed. You can also create image and video files.

General computer-vision and mage-processing algorithms (mid- and low-level APIs)

Using these interfaces, you can experiment with many standard computer vision algorithms without having to code them yourself. These include edge, line, and corner detection, ellipse fitting, image pyramids for multiscale processing, template matching, various transforms (Fourier, discrete cosine, and distance transforms), and more.

High-level computer-vision modules

OpenCV includes several high-level capabilities. In addition to face-detection, recognition, and tracking, it includes optical flow (using camera motion to determine 3D structure), camera calibration, and stereo.

AI and machine-learning methods

Computer-vision applications often require machine learning or other AI methods. Some of these are available in OpenCV's Machine Learning package.

Image sampling and view transformations

It's often useful to process a group of pixels as a unit. OpenCV includes interfaces for extracting image subregions, random sampling, resizing, warping, rotating, and applying perspective effects.

Methods for creating and analyzing binary (two-valued) images

Binary images are frequently used in inspection systems that scan for shape defects or count parts. A binary representation is also convenient when locating an object to grasp.

Methods for computing 3D information

These functions are useful for mapping and localization – either with a stereo rig or with multiple views from a single camera.

Math routines for image processing, computer vision, and image interpretation

OpenCV includes math commonly used, algorithms from linear algebra, statistics, and computational geometry.

Graphics

These interfaces let you write text and draw on images. In addition to various fun and creative possibilities, these functions are useful for labeling and marking. For example, if you write a program that detects objects, it's helpful to label images with their sizes and locations.

GUI methods

OpenCV includes its own windowing interfaces. While these are limited compared to what can be done on each platform, they provide a simple, multi-platform API to display images, accept user input via mouse or keyboard, and implement slider controls.

Datastructures and algorithms

With these interfaces, you can efficiently store, search, save, and manipulate large lists, collections (also called sets), graphs, and trees.

Data persistence

These methods provide convenient interfaces for storing various types of data to disk and retrieving them later.

Figure 1 shows a few examples of OpenCV's capabilities in action: face detection, contour detection, and edge detection.

Organization

OpenCV's functionality is contained in several modules.

CXCORE contains basic datatype definitions. For example, the data structures for image, point, and rectangle are defined in cxtypes.h. CXCORE also contains linear algebra and statistics methods, the persistence functions, and error handlers. Somewhat oddly, the graphics functions for drawing on images are located here, as well.

CV contains image processing and camera calibration methods. The computational geometry functions are also located here.

CVAUX is described in OpenCV's documentation as containing obsolete and experimental code. However, the simplest interfaces for face recognition are in this module. The code behind them is specialized for face recognition, and they're widely used for that purpose.

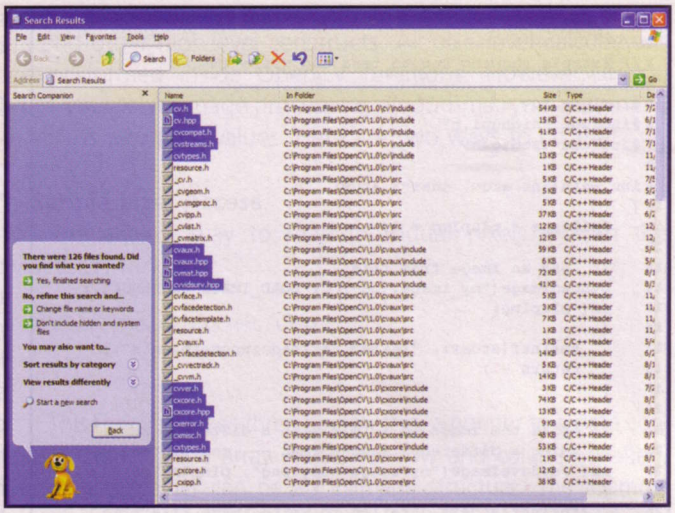

FIGURE 2. Selecting OpenCV header files in Windows to place into a single include directory.

ML contains machine-learning interfaces

The remaining functionality is contained in HighGUI and CVCAM. Both of these are located in a directory named "otherlibs," making them easy to miss. Since HighGUI contains the basic I/O interfaces, you'll want to be sure you don't overlook it! It also contains the multi-platform windowing capabilities.

CVCAM contains interfaces for video access through DirectX on 32-bit Windows platforms. However, HighGUI also contains video interfaces. In this article, I'll cover only the interfaces in HighGUI. They're simpler to use, and they work on all platforms. If you're using Windows XP or 2000, you may get a performance boost by switching to the CVCAM interfaces, but for learning OpenCV, the simpler ones in HighGUI are just fine.

Installing OpenCV

Basic Install

OpenCV for Linux or MacOSX is packaged as a source-code archive. You'll need to build both static and shared-object libraries. You can either build an RPM first and install from that, or compile and install it directly. Instructions for doing both are in INSTALL.

The Windows download is packaged as an executable that installs OpenCV when you run it. It places OpenCV files into a directory of your choice, optionally modifies your system path to include the OpenCV binaries, and registers several DirectX filters. By default, it installs to C:/Program Files/OpenCV/<version>.

Customizing a Windows Install

For Windows users, OpenCV is easy to install, and the default installation will work. But a bit of advance planning may leave you happier with the results. Here are a few suggestions.

Since OpenCV is a developers' toolkit — not a program — you may want to locate it somewhere other than your Program Files directory. If you do prefer to locate it elsewhere, decide that before you run the installer, and enter that location when asked.

I suggest you also decide — before installing — how you want Windows to find the OpenCV dlls. You can either modify your system's PATH variable to include their location, or you can move them, after installing, from OpenCV's "bin" directory to your SYSTEM_ROOT directory.

If you prefer to move the dlls, but aren't sure where your SYSTEM_ROOT directory is, you can locate it by running the sysinfo utility available at **www.cognotics.com/utilities**.

If you prefer to modify the PATH rather than moving the dlls, you can have the installer do that for you by selecting the check box "Add bin directory to PATH."

After Installing

The OpenCV directory contains several subdirectories. The docs directory contains html documentation for all the OpenCV functions and datatypes. Since the best documentation is a working example, you might also want to browse the "samples" directory.

The header files you'll need to include when you compile programs that use OpenCV are distributed among the

```
1 // ImageIO.c
2 //
3 // Example showing how to read and write images
4
5 #include "cv.h"
6 #include "highgui.h"
7 #include <stdio.h>
8
9 int main(int argc, char** argv)
10 {
11    IplImage * pInpImg = 0;
12
13    // Load an image from file
14    cvLoadImage("my_image.jpg", CV_LOAD_IMAGE_UNCHANGED);
15    if(!pInpImg)
16    {
17       fprintf(stderr, "failed to load input image\n");
18       return -1;
19    }
20
21    // Write the image to a file with a different name,
22    // using a different image format -- .png instead of .jpg
23    if( !cvSaveImage("my_image_copy.png", pInpImg) )
24    {
25       fprintf(stderr, "failed to write image file\n");
26    }
27
28    // Remember to free image memory after using it!
29    cvReleaseImage(&pInpImg);
30
31    return 0;
32 }
```

FIGURE 3. Example program that reads an image from a file and writes it to a second file in a different compression format.

OpenCV modules. Although you don't need to do this, I like to gather them together into a single include directory.

On both Linux and Windows, you can locate the headers by searching the install directory and subdirectories for filenames that match the pattern *.h, *.hpp. There will be lots of matches. You don't need all of them. Headers for all modules except HighGUI are in separate "include" directories inside each module. You can skip headers in the "src" directories for these modules. For HighGUI, you'll need highgui.h, located in otherlibs/highgui.

Programming with OpenCV: Some Basics

More about Headers and Libraries

Most OpenCV programs need to include cv.h and highgui.h. Later, for face recognition, we'll also include cvaux.h. The remaining header files are included by these top-level headers.

If you've left the header files in multiple directories (default installation), make sure your compiler's include path contains these directories. If you've gathered the headers into one include directory, make sure that directory is on your compiler's include path.

Your linker will need both the library path and the names of the static libraries you need to link to are cxcore.lib, cv.lib, and highgui.lib. Later, for face recognition, you'll also link to cvaux.lib. These are in OpenCV's "lib" directory.

```
1 // Capture.c
2 //
3 // Example showing how to connect to a webcam and capture
4 // video frames
5
6 #include "stdio.h"
7 #include "string.h"
8 #include "cv.h"
9 #include "highgui.h"
10
11 int main(int argc, char ** argv)
12 {
13    CvCapture * pCapture    = 0;
14    IplImage *  pVideoFrame = 0;
15    int         i;
16    char        filename[50];
17
18    // Initialize video capture
19    pCapture = cvCaptureFromCAM( CV_CAP_ANY );
20    if( !pCapture )
21    {
22       fprintf(stderr, "failed to initialize video capture\n");
23       return -1;
24    }
25
26    // Capture three video frames and write them as files
27    for(i=0; i<3; i++)
28    {
29       pVideoFrame = cvQueryFrame( pCapture );
30       if( !pVideoFrame )
31       {
32          fprintf(stderr, "failed to get a video frame\n");
33       }
34
35       // Write the captured video frame as an image file
36       sprintf(filename, "VideoFrame%d.jpg", i+1);
37       if( !cvSaveImage(filename, pVideoFrame) )
38       {
39          fprintf(stderr, "failed to write image file %s\n", filename);
40       }
41
42       // IMPORTANT: Don't release or modify the image returned
43       // from cvQueryFrame() !
44    }
45
46    // Terminate video capture and free capture resources
47    cvReleaseCapture( &pCapture );
48
49    return 0;
50 }
```

FIGURE 4. Example program that captures live video frames and stores them as files.

Reading and Writing Images

Image I/O is easy with OpenCV. Figure 3 shows a complete program listing for reading an image from file and writing it as a second file, in a different compression format.

To read an image file, simply call cvLoadImage(), passing it the filename (line 14). OpenCV supports most common image formats, including JPEG, PNG, and BMP. You don't need to provide format information. cvLoadImage() determines file format by reading the file header.

To write an image to file, call cvSaveImage(). This function decides which file format to use from the file extension. In this example, the extension is "png," so it will write the image data in PNG format.

Both cvLoadImage() and cvSaveImage() are in the HighGUI module.

When you're finished using the input image received from cvLoadImage(), free it by calling cvReleaseImage(), as on line 29. This function takes an address of a pointer as its input because it does a "safe release." It frees the image structure only if it's non-null. After freeing it, it sets the image pointer to 0.

Live Video Input

Capturing image frames from a webcam, or other digital video device, is nearly as easy as loading from file. Figure 4 shows a complete program listing to initialize frame capture, capture and store several video frames, and close the capture interface.

The capture interface is initialized, on line 19, by calling cvCaptureFromCAM(). This function returns a pointer to a CvCapture structure. You won't access this structure directly. Instead, you'll store the pointer to pass to cvQueryFrame().

When you're finished using video input, call cvReleaseCapture() to release video resources. As with cvReleaseImage(), you pass the address of the CvCapture pointer to cvReleaseCapture().

Don't release or otherwise modify

the IplImage you receive from cvQueryFrame()! If you need to modify image data, create a copy to work with:

```
// Copy the video frame
IplImage *pImgToChange =
    cvCloneImage(pVideoFrame);

// Insert your image-processing code here ...

// Free the copy after using it
cvReleaseImage(&pImgToChange);
```

Color Conversions

Figure 5 shows code for converting a color image to grayscale. OpenCV has built-in support for converting to and from many useful color models, including RGB, HSV, YCrCb, and CIELAB. (For a discussion of color models, see "The World of Color," *SERVO Magazine*, November 2005.)

Note that the conversion function, cvCvtColor(), requires two images in its input list. The first one, pRGBImg, is the source image. The second, pGrayImg, is the destination image. It will contain the conversion result when cvCvtColor() returns.

Because this paradigm of passing source and destination images to a processing function is common in OpenCV, you'll frequently need to create a destination image. On line 25, a call to cvCreateImage() creates an image the same size as the original, with uninitialized pixel data.

How OpenCV Stores Images

OpenCV stores images as a C structure, IplImage. IPL stands for Image Processing Library, a legacy from the original OpenCV versions that required this product.

The IplImage datatype is defined in CXCORE. In addition to raw pixel data, it contains a number of descriptive fields, collectively called the Image Header. These include

- Width — Image width in pixels
- Height — Image height in pixels
- Depth — One of several predefined constants that indicate the number of bits per pixel per channel. For example, if depth=IPL_DEPTH_8U, data for each pixel channel are stored as eight-bit, unsigned values.
- nChannels — The number of data channels (from one to four). Each channel contains one type of pixel data. For example, RGB images have three channels – red, green, and blue intensities. (These are sometimes called BGR images, because pixel data are stored as blue, green, then red values.) Grayscale images contain only one channel – pixel brightness.

Accessing Pixel Values

It's possible to create many types of functionality using OpenCV without directly accessing raw pixel data. For example, the face detection, tracking, and recognition programs described later in this series never manipulate raw pixel data directly. Instead, they work with image pointers and other high-level constructs. All pixel-level calculations are performed inside OpenCV functions. However, if you write your own image-processing algorithms, you may need to access raw pixel values. Here are two ways to do that:

1. Simple Pixel Access

The easiest way to read individual pixels is with the cvGet2D() function:

```
CvScalar cvGet2D(const CvArr*,
                 int row, int col);
```

This function takes three parameters: a pointer to a data container (CVArr*), and array indices for row and column location. The data container can be an IplImage structure. The topmost row of pixels is row=0, and the bottommost is row=height-1.

The cvGet2D()function returns a C structure, CvScalar, defined as

```
typedef struct CvScalar
{
            double val[4];
}
CvScalar;
```

The pixel values for each channel are in val[i]. For grayscale images, val[0] contains pixel brightness. The other three values are set to 0. For a three-channel, BGR image, blue=val[0], green=val[1], and red=val[2].

The complementary function, cvSet2D(), allows you to modify pixel values. It's defined as

```
1  // ConvertToGray.c
2  //
3  // Example showing how to convert an image from color
4  // to grayscale
5
6  #include "stdio.h"
7  #include "string.h"
8  #include "cv.h"
9  #include "highgui.h"
10
11 int main(int argc, char** argv)
12 {
13    IplImage * pRGBImg  = 0;
14    IplImage * pGrayImg = 0;
15
16    // Load the RGB image from file
17    pRGBImg = cvLoadImage("my_image.jpg", CV_LOAD_IMAGE_UNCHANGED);
18    if(!pRGBImg)
19    {
20       fprintf(stderr, "failed to load input image\n");
21       return -1;
22    }
23
24    // Allocate the grayscale image
25    pGrayImg = cvCreateImage
26       ( cvSize(pRGBImg->width, pRGBImg->height), pRGBImg->depth, 1 );
27
28    // Convert it to grayscale
29    cvCvtColor(pRGBImg, pGrayImg, CV_RGB2GRAY);
30
31    // Write the grayscale image to a file
32    if( !cvSaveImage("my_image_gray.jpg", pGrayImg) )
33    {
34       fprintf(stderr, "failed to write image file\n");
35    }
36
37    // Free image memory
38    cvReleaseImage(&pRGBImg);
39    cvReleaseImage(&pGrayImg);
40
41    return 0;
42 }
```

FIGURE 5. Example program for converting a color image to grayscale.

Resources

Sourceforge site
http://sourceforge.net/projects/opencvlibrary

Official OpenCV usergroup
http://tech.groups.yahoo.com/group/OpenCV

OpenCV Wiki
http://opencvlibrary.sourceforge.net

Source code for the program listings in this article are available for download at **www.cognotics.com/opencv/servo**.

```
void cvSet2D(CvArr*, int row, int col,
             CvScalar);
```

2. Fast Pixel Access

Although cvGet2D() and cvSet2D() are easy to use, if you want to access more than a few pixel values, and performance matters, you'll want to read values directly from the raw data buffer, IplImage.imageData.

Image data in the buffer are stored as a 1D array, in row-major order. That is, all pixel values in the first row are listed first, followed by pixel values in the second row, and so on.

For performance reasons, pixel data are aligned, and padded if necessary, so that each row starts on an even four-byte multiple. A second field, IplImage.widthStep, indicates the number of bytes between the start of each row's pixel data. That is, row i starts at IplImage.imageData + i*IplImage.widthStep.

IplImage.imageData is defined as type char*, so you may need to cast the data type. For example, if your image data are unsigned bytes (the most common input type), you'd cast each value to unsigned char* before assigning, or otherwise using, it.

If you're accessing data from a grayscale (single-channel) image, and the data depth is eight bits (one byte per pixel), you'd access pixel[row][col] with

```
pixel[row][col] = ((uchar*)
  (pImg->imageData +
  row*pImg->widthStep + col));
```

In multi-channel images, channel values are interlaced. Here's a code snippet to access blue, green, and red pixel values:

```
step  = pImg->widthStep;
nChan = pImg->nChannels;
// = 3 for a BGR image
buf   = pImg->imageData;

blue[row][col] =
                ((uchar*)(buf + row*widthStep +
                nChan*col);
green[row][col] =
                ((uchar*)(buf + row*widthStep +
                nChan*col + 1);
red[row][col] =
                ((uchar*)(buf + row*widthStep +
                nChan*col + 2);
```

Finally, if image depth is greater than eight bits (for example, IPL_DEPTH_32S), you'd need to transfer multiple bytes for each value and multiply the buffer offset by the number of data bytes for your image depth. It's very unlikely, however, that you'll encounter a situation in which you must access multi-byte pixel values directly.

Finding Help

If you have problems installing or using OpenCV, the first place to turn for help is the FAQ (faq.htm) in your OpenCV docs directory. The INSTALL file, at the root of your OpenCV directory, also contains helpful setup and troubleshooting tips. If these don't answer your question, you may want to post a query to the official Yahoo! user group. The group's URL is in the Resources sidebar.

API documentation for each module is in the docs/ref subdirectory. All reference manuals except the one for CVAUX are linked from index.htm, in the docs directory.

Coming Up ...

Next month, I'll show you how to detect faces with OpenCV and explain the algorithm behind the interface. Be seeing you! SV

About the Author

Robin Hewitt is an independent software consultant working in the areas of computer vision and robotics. She has worked as a Computer Vision Algorithm Developer at Evolution Robotics and is a member of SO(3), a computer-vision research group at UC San Diego. She is one of the original developers of SodaVision, an experimental face-recognition system at UC San Diego. SodaVision was built with OpenCV.