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.

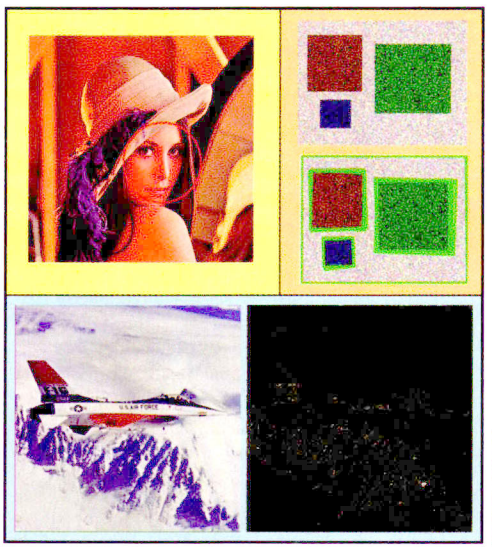


FIGURE 1. Among OpenCV's many capabilities are face detection (top left), contour detection (top right), and edge detection (bottom).

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

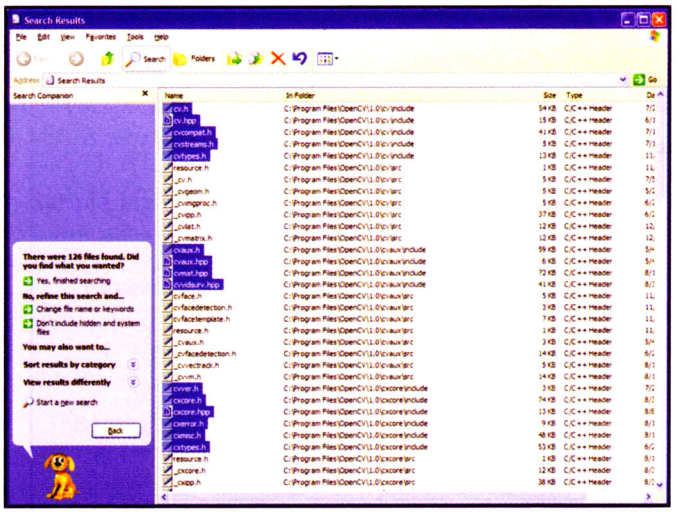
OpenCV's functionality is contained in several modules.

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.

Seeing With OpenCV

Finding Faces in Images

by Robin Hewitt

PART 2

Last month's article in this series introduced OpenCV – Intel's free, open-source computer vision library for C/C++ programmers. It covered the basics – downloading and installing OpenCV, reading and writing image files, capturing video, and working with the IplImage data structure.

This month, I'll show you how to use OpenCV to detect faces. I'll explain how the face detection algorithm works, and give you tips for getting the most out of it.

Background and Preliminaries

OpenCV uses a type of face detector called a Haar Cascade classifier. The sidebar, "How Face Detection Works, or What's a Haar Cascade Classifier, Anyhow?" explains what this mouthful means. Figure 1 shows an example of OpenCV's face detector in action.

Given an image – which can come from a file or from live video – the face detector examines each image location and classifies it as "Face" or "Not Face." Classification assumes a fixed scale for the face, say 50 x 50 pixels. Since faces in an image might be smaller or larger than this, the classifier runs over the image several times, to search for faces across a range of scales. This may seem like an enormous amount of processing, but thanks to algorithmic tricks (explained in the sidebar), classification is very fast, even when it's applied at several scales.

The classifier uses data stored in an XML file to decide how to classify each image location. The OpenCV download includes four flavors of XML data for frontal face detection, and one for profile faces. It also includes three non-face XML files: one for full body (pedestrian) detection, one for upper body, and one for lower body.

You'll need to tell the classifier where to find the data file you want it to use. The one I'll be using is called haarcascade_frontal face_default.xml. In OpenCV version 1.0, it's located at:

```
[OPENCV_ROOT]/data/haarcascades/
haarcascade_frontalface_default.
xml
```

where `[OPENCV_ROOT]` is the path to your OpenCV installation. For example, if you're on Windows XP and you selected the default installation location, you'd use:

```
[OPENCV_ROOT] = "C:/Program Files/
OpenCV"
```

(If you're working with an older, 16-bit version of Windows, you'd use '\' as the directory separator, instead of '/'.)

It's a good idea to locate the XML file you want to use and make sure your path to it is correct before you code the rest of your face detection program.

You'll also need an image to process. The image lena.jpg is a good one to test with. It's located in the OpenCV samples/c directory. If you copy it to your program's working directory, you'll easily be able to compare your program's output with the output from the code in Figure 2.

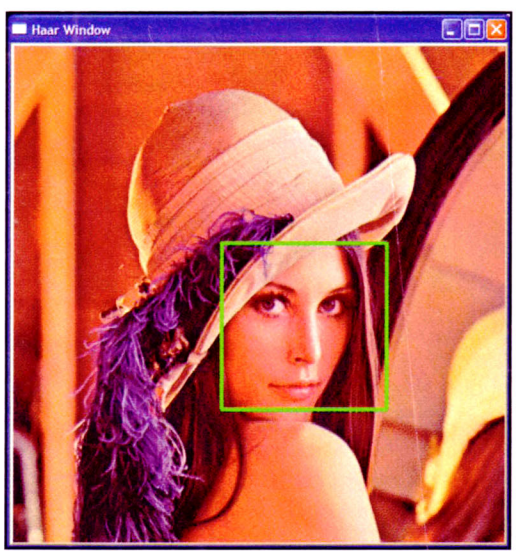


FIGURE 1. Face detection with OpenCV, using default parameters. The input image is lena.jpg, in the samples/c directory.

Implementing Face Detection, Step by Step

Figure 2 shows the source code to load an image from a file, detect faces in it, and display the image with detected faces outlined in green. Figure 1 shows the display produced by this program when it's run from the command line, using:

```
DetectFaces lena.jpg
```

Initializing (and running) the Detector

The variable `CvHaarClassifier Cascade * pCascade` (Line 2) holds the data from the XML file you located earlier. To load the XML data into `pCascade`, you can use the `cvLoad()` function, as in Lines 11-13. `cvLoad()` is a general-purpose function for loading data from files. It takes up to

FIGURE 2. Source code to detect faces in one image. Usage: Detect Faces <image file>.

```
1 // declarations
2 CvHaarClassifierCascade * pCascade = 0;       // the face detector
3 CvMemStorage * pStorage = 0;                  // expandable memory buffer
4 CvSeq * pFaceRectSeq;                         // list of detected faces
5 int i;
6
7 // initializations
8 IplImage * pInpImg = (argc > 1) ?
9    cvLoadImage(argv[1], CV_LOAD_IMAGE_COLOR) : 0;
10 pStorage = cvCreateMemStorage(0);
11 pCascade = (CvHaarClassifierCascade *)cvLoad
12   ((OPENCV_ROOT"/data/haarcascades/haarcascade_frontalface_default.xml"),
13   0, 0, 0 );
14
15 // validate that everything initialized properly
16 if( !pInpImg || !pStorage || !pCascade )
17 {
18    printf("Initialization failed: %s \n",
19      (!pInpImg)?     "didn't load image file" :
20      (!pCascade)?    "didn't load Haar cascade -- "
21                      "make sure path is correct" :
22      "failed to allocate memory for data storage");
23    exit(-1);
24 }
25
26 // detect faces in image
27 pFaceRectSeq = cvHaarDetectObjects
28    (pInpImg, pCascade, pStorage,
29    1.1,                          // increase search scale by 10% each pass
30    3,                            // drop groups of fewer than three detections
31    CV_HAAR_DO_CANNY_PRUNING,     // skip regions unlikely to contain a face
32    cvSize(0,0));                 // use XML default for smallest search scale
33
34 // create a window to display detected faces
35 cvNamedWindow("Haar Window", CV_WINDOW_AUTOSIZE);
36
37 // draw a rectangular outline around each detection
38 for(i=0;i<(pFaceRectSeq? pFaceRectSeq->total:0); i++ )
39 {
40    CvRect * r = (CvRect*)cvGetSeqElem(pFaceRectSeq, i);
41    CvPoint pt1 = { r->x, r->y };
42    CvPoint pt2 = { r->x + r->width, r->y + r->height };
43    cvRectangle(pInpImg, pt1, pt2, CV_RGB(0,255,0), 3, 4, 0);
44 }
45
46 // display face detections
47 cvShowImage("Haar Window", pInpImg);
48 cvWaitKey(0);
49 cvDestroyWindow("Haar Window");
50
51 // clean up and release resources
52 cvReleaseImage(&pInpImg);
53 if(pCascade) cvReleaseHaarClassifierCascade(&pCascade);
54 if(pStorage) cvReleaseMemStorage(&pStorage);
```

three input parameters. For this example, you'll only need the first parameter. This is the path to an XML file containing a valid Haar Cascade. Here, I've loaded the default frontal face detector included with OpenCV. If you're coding in C, set the remaining parameters to 0. If you're coding in C++, you can simply omit the unused parameters from your function call.

Before detecting faces in images, you'll also need to instantiate a `CvMemStorage` object (`pStorage`, declared at Line 3). This is a memory buffer that expands automatically, as needed. The face detector will put the list of detected faces into this buffer. Since the buffer is expandable, you won't need to worry about overflowing it. All you'll have to do is create it (Line 10), then release it when you're finished (Line 54).

You'll often need to load data from files with OpenCV. Since it's easy to get a path wrong, it's a good idea to insert a quick check to make sure everything loaded and initialized properly. Lines 16-24 do a simple error check, print a diagnostic message, and exit if initialization fails.

Running the Detector

Lines 27-32 call `cvHaarDetectObjects()` to run the face detector. This function takes up to seven parameters. The first three are the image pointer, XML data, and memory buffer. The remaining four parameters are set to their C++ defaults. These last four parameters are described below, in the section, "Parameters and Tuning."

Viewing the Results

A quick way to check if your program works is to display the results in an OpenCV window. You can create a display window using the `cvNamedWindow()` function, as in Line 35. The first parameter is a string, with a window name. The second, `CV_WINDOW_AUTOSIZE`, is a flag that tells the window to automatically resize itself to fit the image you give it to display.

To pass an image for display, call `cvShowImage()` with the name you previously assigned the window, and the image you want it to display. The `cvWaitKey()` call at Line 48 pauses the application until you close the window. If the window fails to close by clicking its close icon, click inside the window's display area, then press a keyboard key. Also, make sure your program calls `cvDestroyWindow()` (Line 49) to close the window.

Face detections are stored as a list of `CvRect struct` pointers. Lines 38-44 access each detection rectangle and add its outline to the `pInpImg` variable, which holds the in-memory image loaded from the file.

Releasing Resources

Lines 52-54 release the resources used by the input image, the XML data, and the storage buffer. If you'll be detecting faces in multiple images, you don't need to release the

How Face Detection Works

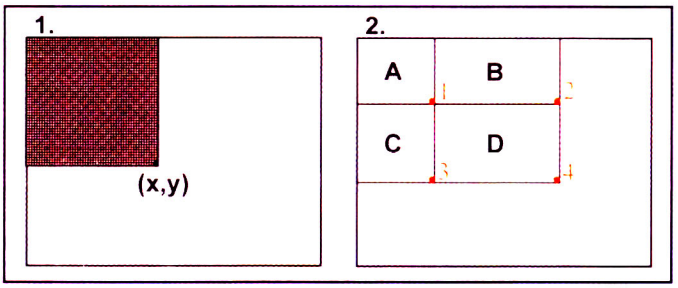


FIGURE B. The Integral Image trick. After integrating, the pixel at (x,y) contains the sum of all pixel values in the shaded rectangle. The sum of pixel values in rectangle D is (x4, y4) – (x2, y2) – (x3, y3) + (x1, y1).

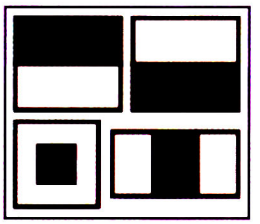


FIGURE A. Examples of the Haar features used in OpenCV.

OpenCV's face detector uses a method that Paul Viola and Michael Jones published in 2001. Usually called simply the Viola-Jones method, or even just Viola-Jones, this approach to detecting objects in images combines four key concepts:

- Simple rectangular features, called Haar features.
- An Integral Image for rapid feature detection.
- The AdaBoost machine-learning method.
- A cascaded classifier to combine many features efficiently.

The features that Viola and Jones used are based on Haar wavelets. Haar wavelets are single-wavelength square waves (one high interval and one low interval). In two dimensions, a square wave is a pair of adjacent rectangles — one light and one dark.

The actual rectangle combinations used for visual object detection are not true Haar wavelets. Instead, they contain rectangle combinations better suited to visual recognition tasks. Because of that difference, these features are called Haar features, or Haarlike features, rather than Haar wavelets. Figure A shows the features that OpenCV uses.

The presence of a Haar feature is determined by subtracting the average dark-region pixel value from the average light-region pixel value. If the difference is above a threshold (set during learning), that feature is said to be present.

To determine the presence or absence of hundreds of Haar features at every image location and at several scales efficiently, Viola and Jones used a technique called an Integral Image. In general, "integrating" means adding small units together. In this case, the small units are pixel values. The integral value for each pixel is the sum of all the pixels above it and to its left. Starting at the top left and traversing to the right and down, the entire image can be integrated with a few integer operations per pixel.

As Figure B1 shows, after integration, the value at each pixel location, (x,y) contains the sum of all pixel values within a rectangular region that has one corner at the top left of the image and the other at location (x,y). To find the average pixel value in this rectangle, you'd only need to divide the value at (x,y) by the rectangle's area.

But what if you want to know the summed values for some other rectangle, one that doesn't have one corner at the upper left of the image? Figure B2 shows the solution to that problem. Suppose you want the summed values in D. You can think of that as being the sum of pixel values in the combined rectangle, A+B +C+D, minus the sums in rectangles A+B and A+C, plus the sum of pixel values in A. In other words,

$$D = A+B+C+D - (A+B) - (A+C) + A.$$

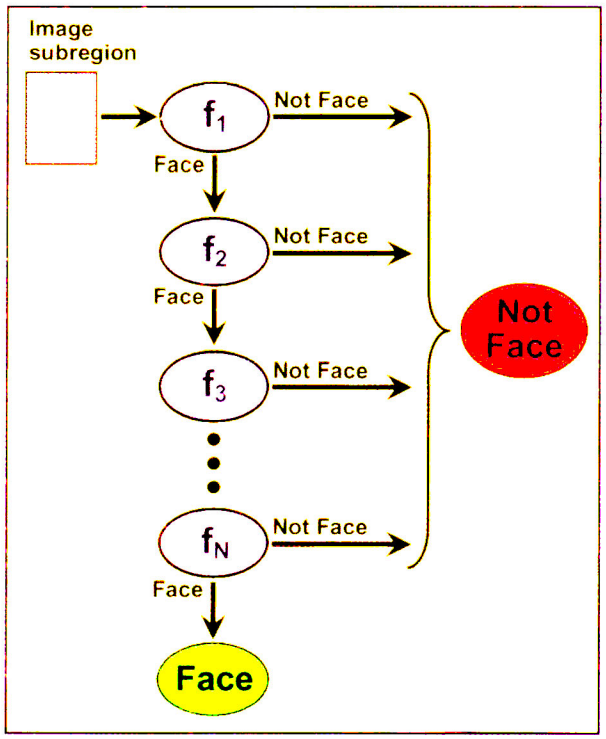
Conveniently, A+B+C+D is the Integral Image's value at location 4, A+B is the value at location 2, A+C is the value at location 3, and A is the value at location 1. So, with an Integral Image, you can find the sum of pixel values for any rectangle in the original image with just three integer operations:

$$(x4, y4) - (x2, y2) - (x3, y3) + (x1, y1).$$

To select the specific Haar features to use and to set threshold levels, Viola and Jones use a machine-learning method called AdaBoost. AdaBoost combines many "weak" classifiers to create one "strong" classifier. "Weak" here means the classifier only gets the right answer a little more often than random guessing would. That's not very good. But if you had a whole lot of these weak classifiers and each one "pushed" the final answer a little bit in the right direction, you'd have a strong, combined force for arriving at the correct solution. AdaBoost selects a set of weak classifiers to combine and assigns a weight to each. This weighted combination is the strong classifier.

Viola and Jones combined weak classifiers as a filter chain, shown in Figure C, that's especial-

FIGURE C. The classifier cascade is a chain of single-feature filters. Image subregions that make it through the entire cascade are classified as "Face." All others are classified as "Not Face."



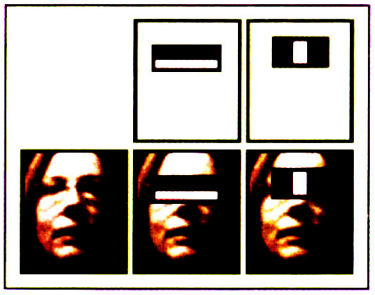


FIGURE D. The first two Haar features in the original Viola-Jones cascade.

ly efficient for classifying image regions. Each filter is a weak classifier consisting of one Haar feature. The threshold for each filter is set low enough that it passes all, or nearly all, face examples in the training set. (The training set is a large database of faces, maybe a thousand or so.) During use, if any one of these filters fails to pass an image region, that region is immediately classified as "Not Face." When a filter passes an image region, it goes to the next filter in the chain. Image regions that pass through all filters in the chain are classified as "Face." Viola and Jones dubbed this filtering chain a cascade.

The order of filters in the cascade is determined by weights that AdaBoost assigns. The more heavily weighted filters come first, to eliminate non-face image regions as quickly as possible. Figure D shows the first two features from the original Viola-Jones cascade superimposed on my face. The first one keys off the cheek area being lighter than the eye region. The second uses the fact that the bridge of the nose is lighter than the eyes.

XML data or the buffer until after you're done detecting faces.

Parameters and Tuning

There are several parameters you can adjust to tune the face detector for your application.

Minimum Detection Scale

The seventh parameter in the call to `cvHaarDetectObjects()` is the size of the smallest face to search for. In C, you can select the default for this by setting the scale to 0x0, as in Figure 2, Line 32. (In C++, simply omit this parameter to use the default.) But what is the default? You can find out by opening the XML file you'll be using. Look for the `<size>` tag. In the default frontal face detector, it's:

```
<size> 24 24 </size>.
```

So, for this cascade, the default minimum scale is 24 x 24.

Depending on the resolution you're using, this default size may be a very small portion of your overall image. A face image this small may not be meaningful or useful, and detecting it takes up CPU cycles you could use for other purposes. For these reasons — and also to minimize the number of face detections your own code needs to process — it's best to set the minimum detection scale only as small as you truly need.

To set the minimum scale higher than the default value, set this parameter to the size you want. A good rule of thumb is to use some fraction of your input image's width or height as the minimum scale — for example, 1/4 of the image width. If you specify a minimum scale other than the default, be sure its aspect ratio (the ratio of width to height) is the same as the default's. In this case, aspect ratio is 1:1.

Minimum Neighbors Threshold

One of the things that happens "behind the scenes" when you call the face detector is that each positive face region actually generates many hits from the Haar detector. Figure 3 shows OpenCV's internal rectangle list for the example image, lena.jpg. The face region itself generates the largest cluster of rectangles. These largely overlap. In addition, there's one small detection to the (viewer's) left, and two larger detections slightly above and left of the main face cluster.

Usually, isolated detections are false detections, so it makes sense to discard these. It also makes sense to somehow merge the multiple detections for each face region into a single detection. OpenCV does both these before returning its list of detected faces. The merge step first groups rectangles that contain a large amount of overlap, then finds the average rectangle for the group. It then replaces all rectangles in the group with the average rectangle.

Between isolated rectangles and large groupings are smaller groupings that may be faces, or may be false detections. The minimum-neighbors threshold sets the cutoff level for discarding or keeping rectangle groups based on how many raw detections are in the group. The C++ default for this parameter is three, which means to merge groups of three or more and discard groups with fewer rectangles. If you find that your face detector is missing a lot of faces, you might try lowering this threshold to two or one.

If you set it to 0, OpenCV will return the complete list of raw detections from the Haar classifier. While you're tuning your face detector, it's helpful to do this just to see what's going on inside OpenCV. Viewing the raw detections will improve your intuition about the effects of changing other parameters, which will help you tune them.

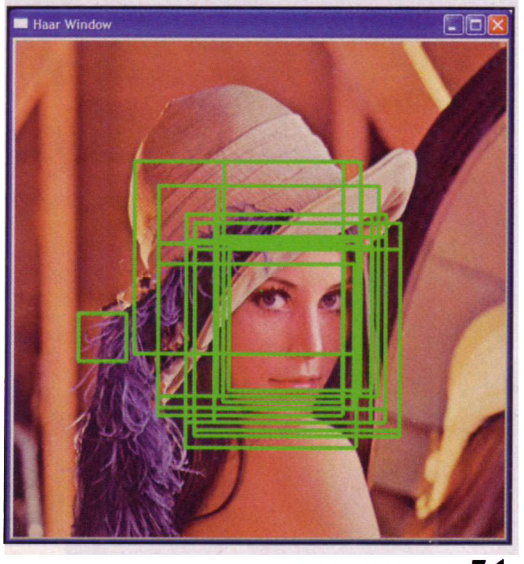


FIGURE 3. OpenCV's internal detection rectangles. To see these, use min_neighbors = 0.

Scale Increase Rate

The fourth input parameter to `cvHaarDetectObjects()` specifies how quickly OpenCV should increase the scale for face detections with each pass it makes over an image. Setting this higher makes the detector run faster (by running fewer passes), but if it's too high, you may jump too quickly between scales and miss faces. The default in OpenCV is 1.1, in other words, scale increases by a factor of 1.1 (10%) each pass.

Canny Pruning Flag

The sixth parameter to `cvHaarDetectObjects()` is a flag variable. There are currently only two options: 0 or `CV_HAAR_DO_CANNY_PRUNING`. If the Canny Pruning option is selected, the detector skips image regions that are unlikely to contain a face, reducing computational overhead and possibly

References and Resources

- *OpenCV on Sourceforge* **http://sourceforge.net/projects/opencvlibrary**
- *Official OpenCV usergroup* **http://tech.groups.yahoo.com/group/OpenCV**
- G. Bradski, A. Kaehler, and V. Pisarevsky, "Learning-Based Computer Vision with Intel's Open Source Computer Vision Library," *Intel Technology Journal*, Vol 9(1), May 19, 2005. **www.intel.com/technology/itj/2005/volume09issue02/art03_learning_vision/p01_abstract.htm**
- R.E. Schapire, "A Brief Introduction to Boosting," *Joint Conference on Artificial Intelligence*, Morgan Kauman, San Francisco, pp. 1401-1406, 1999.
- P. Viola and M.J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," CVPR, 2001.

eliminating some false detections. The regions to skip are identified by running an edge detector (the Canny edge detector) over the image before running the face detector.

Again, the choice of whether or not to set this flag is a tradeoff between speed and detecting more faces. Setting this flag speeds processing, but may cause you to miss some faces. In general, you can do well with it set, but if you're having difficulty detecting faces, clearing this flag may allow you to detect more reliably. Setting the minimum-neighbors threshold to 0 so you can view the raw detections will help you better gauge the effect of using Canny Pruning.

The Haar Cascade

There are several frontal face detector cascades in OpenCV. The best choice for you will depend on your set-up. It's easy to switch between them – just change the file name. Why not try each?

It's also possible to create your own, custom XML file using the HaarTraining application, in OpenCV's apps directory. Using that application is beyond the scope of this article. However, the instructions are in OpenCV's apps/haartraining/docs directory.

Coming Up ...

Now that you've found a face, you might want to follow it around. Next month, I'll show you how to use Camshift, OpenCV's face tracking method, to do just that. 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.

Seeing With OpenCV

Follow That Face!

by Robin Hewitt

PART 3

Last month's article in this series explained how to implement and configure face detection. This month, I'll show you how to use OpenCV to track a face once you've detected it.

Face Tracking in OpenCV

Tracking a face is more difficult than tracking a strongly-colored object. Skin reflects the ambient light in subtle, changing ways as a person's head turns or tilts.

In principle, you could track a face by locating it over and over in every frame, using the Haar detector described in last month's article. To do that, however, you'd need to decide if the face you detected in each frame is the same face. If the detector finds more than one face in a frame, you'd need to decide which detection is the one you're tracking. Finally, if a person's head tilts towards one shoulder, or turns towards profile view, the frontal face detector will no longer detect it, so you'd need to handle that situation, as well.

Fortunately, OpenCV includes specialized code for tracking a face efficiently, using continuity between frames to help find the best match for the face it's following.

The algorithm that OpenCV uses for face tracking is called Camshift. Camshift uses color information, but rather than relying on a single color, it tracks a combination of colors. Since it tracks by color, it can follow a face through orientation changes that the Haar detector can't handle. The sidebar, "How OpenCV's Face Tracker Works," explains this algorithm in more detail.

Camshift was originally developed for hands-free gaming. It's designed to be very fast and "lightweight" so the computer can do other tasks while tracking. Since it was developed as a gaming interface, Camshift also has an (limited) ability to detect changes in head position, such as tilting the head to one side. Could you use that ability to communicate with your robot? Maybe two fast head tilts mean "Come here, robot!"

Figure 1 shows OpenCV's face tracker in action – following a face as it tilts to one side and during a turn to profile.

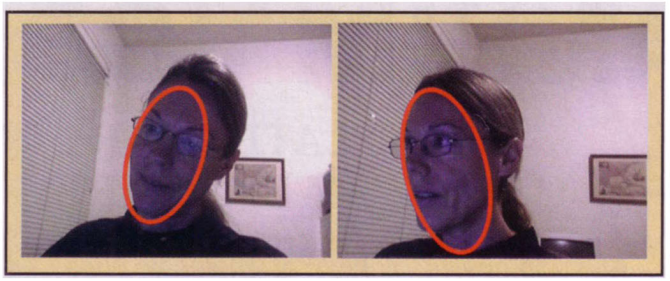


FIGURE 1. OpenCV's face tracker in action. It's able to follow a face as it tilts to one side and during a turn to profile.

The Camshift Demo

The OpenCV samples directory contains a program called camshiftdemo. You can get some good hands-on experience and an intuitive feel for the Camshift algorithm with this demo program. Here are the steps for doing that:

1) Plug in a webcam.

2) Launch the program called camshiftdemo in the samples directory.

3) Use your mouse to select a rectangle centered tightly on your face.

4) Click in the video-display window and type the letter *b*. (The display should change to look something like the view in Figure 2.)

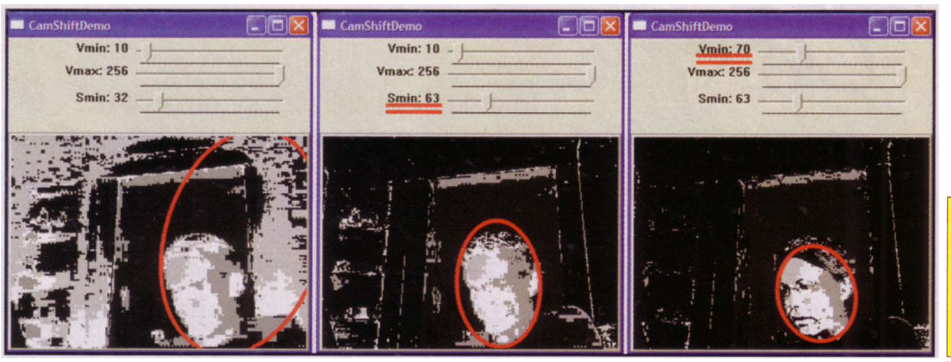


FIGURE 2. To tune the Camshift parameters `smin` and `vmin`, run the camshiftdemo program in the samples directory. These parameters are easier to set if you toggle to the backprojection view by clicking in the view window, then typing *b*.

How OpenCV's Face Tracker Works

OpenCV's face tracker uses an algorithm called Camshift. Camshift consists of four steps:

1) Create a color histogram to represent the face.

2) Calculate a "face probability" for each pixel in the incoming video frames.

3) Shift the location of the face rectangle in each video frame.

4) Calculate the size and angle.

Here's how each step works:

1) *Create a histogram.* Camshift represents the face it's tracking as a histogram (also called a barchart) of color values. Figure A shows two example histograms produced by the Camshift demo program that ships with OpenCV. The height of each colored bar indicates how many pixels in an image region have that "hue." Hue is one of three values describing a pixel's color in the HSV (Hue, Saturation, Value) color model. (For more on color and color models, see "The World of Color," *SERVO Magazine*, November '05.)

In the image region represented by the top histogram, a bluish hue is most common, and a slightly more lavender hue is the next most common. The bottom histogram shows a region in which the most common hue is the rightmost bin. This hue is almost, but not quite, red.

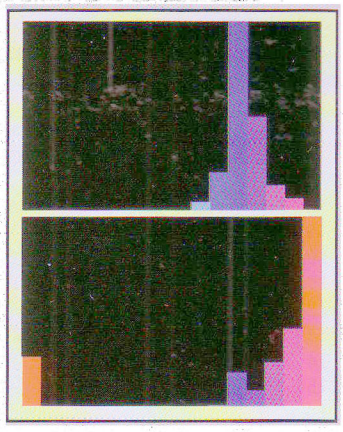


FIGURE A. Two examples of the color histogram that Camshift uses to represent a face.

2) *Calculate face probability — simpler than it sounds!* The histogram is created only once, at the start of tracking. Afterwards, it's used to assign a "face-probability" value to each image pixel in the video frames that follow.

"Face probability" sounds terribly complicated and heavily mathematical, but it's neither! Here's how it works. Figure B shows the bars from a histogram stacked one atop the other. After stacking them, it's clear that the rightmost bar accounts for about 45% of the pixels in the region. That means the probability that a pixel selected randomly from this region would fall into the rightmost bin is 45%. That's the "face probability" for a pixel with this hue. The same reasoning indicates that the face probability for the next histogram bin to the right is about 20%, since it accounts for about 20% of the stack's total height. That's all there is to it.

As new video frames arrive, the hue value for each pixel is determined. From that, the face histogram is used to assign a face probability to the pixel. This process is called "histogram backprojection" in OpenCV. There's a built-in method that implements it, called `cvCalcBackProject()`.

Figure C shows the face-probability image in one video frame as Camshift tracks my face. Black pixels have the lowest probability value, and white, the highest. Gray pixels lie somewhere in the middle.

3) *Shift to a new location.* With each new video frame, Camshift "shifts" its estimate of the face location, keeping it centered over the area with the highest concentration of bright pixels in the face-probability image. It finds this new location by starting at the previous location and computing the center of gravity of the face-probability values within a rectangle. It then shifts the rectangle so it's right over the center of gravity. It does this a few times to center the rectangle well. The OpenCV function `cvCamShift()` implements the steps for shifting to the new location.

This process of shifting the rectangle to correspond with the center of gravity is based on an algorithm called "Mean Shift," by Dorin Comaniciu. In fact, Camshift stands for "Continuously Adaptive Mean Shift."

4) *Calculate size and angle.* The OpenCV method is called "Continuously Adaptive" and not just "Mean Shift" because it also adjusts the size and angle of the face rectangle each time it shifts it. It does this by selecting the scale and orientation that are the best fit to the face-probability pixels inside the new rectangle location.

FIGURE C. The normal and face-probability views as Camshift tracks my face. In the face-probability view, black pixels have the lowest value, and white, the highest. Gray pixels lie somewhere in the middle.

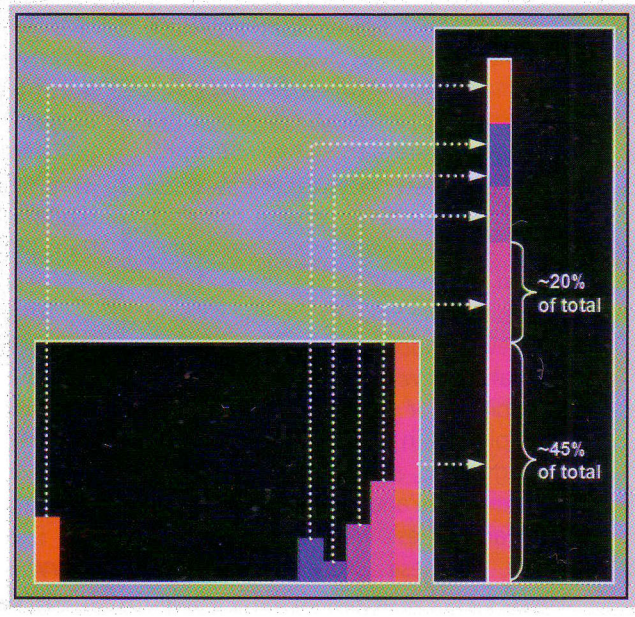


FIGURE B. To see what "face probability" means, imagine stacking the bars in a histogram one atop the other. The probability associated with each color is the percent that color bar contributes to the total height of this stack.

main()

```
1 //// Constants
2 const char * DISPLAY_WINDOW = "DisplayWindow";
3 #define OPENCV_ROOT  "C:/Program Files/OpenCV/1.0"
4
5 //// Global variables
6 IplImage  * pVideoFrameCopy = 0;
7
8 void main( int argc, char** argv )
9 {
10   CvRect * pFaceRect = 0;
11   if( !initAll() ) exitProgram(-1);
12
13   // Capture and display video frames until a face
14   // is detected
15   while( 1 )
16   {
17      // Look for a face in the next video frame
18      captureVideoFrame();
19      pFaceRect = detectFace(pVideoFrameCopy);
20
21      // Show the display image
22      cvShowImage( DISPLAY_WINDOW, pVideoFrameCopy );
23      if( (char)27==cvWaitKey(1) ) exitProgram(0);
24
25      // exit loop when a face is detected
26      if(pFaceRect) break;
27   }
28
29   // initialize tracking
30   startTracking(pVideoFrameCopy, pFaceRect);
31
32   // Track the detected face using CamShift
33   while( 1 )
34   {
35      CvBox2D faceBox;
36
37      // get the next video frame
38      captureVideoFrame();
39
40      // track the face in the new video frame
41      faceBox = track(pVideoFrameCopy);
42
43      // outline face ellipse
44      cvEllipseBox(pVideoFrameCopy, faceBox,
45          CV_RGB(255,0,0), 3, CV_AA, 0 );
46      cvShowImage( DISPLAY_WINDOW, pVideoFrameCopy );
47      if( (char)27==cvWaitKey(1) ) break;
48   }
49
50   exitProgram(0);
51 }
```

FIGURE 3. The main program listing for detecting a face in a live video stream, then tracking it using the Camshift wrapper API.

5) Adjust the sliders for `smin` and `vmin` until the ellipse is well positioned and the background is mostly black.

6) Repeat Step 4 to toggle back to normal view, then use Camshift to track your face.

Tuning Camshift

As mentioned above, Camshift uses a combination of colors to track faces. In the representation that Camshift uses, color is undefined for pixels that have a neutral shade (white, gray, or black). Color can be computed for pixels that are *almost* neutral, but their color values are unstable, and these pixels contribute noise that interferes with tracking.

Camshift uses two parameters — `smin` and `vmin` — to screen out this noise. These parameters define thresholds for ignoring pixels that are too close to neutral. `vmin` sets the threshold for "almost black," and `smin` for "almost gray." These two threshold levels will need to be adjusted for your setup to get good results with Camshift.

Camshift also uses a third parameter called `vmax`, to set a threshold for pixels that are too bright. But `smin` has the side effect of also eliminating pixels that are close to white, so you shouldn't need to tweak `vmax` to get good results.

The easiest way to select good values for your setup is with camshiftdemo. As suggested in the preceding section, it's easier to set these if you toggle the viewing mode by clicking the view window and typing *b*. (This alternative view is the called the "face-probability," or "backprojection" view. It's explained in the sidebar.)

Figure 2 shows the effect of adjusting `smin` and `vmin`. Initially, in the first frame, these were at their default values. At these levels, Camshift displayed a very large ellipse that included not only my face, but half the room as well! The reason for the oversized face detection is clearly visible in the face-probability view. Background pixels with a nearly neutral shade contributed too much noise when `vmin` and `smin` were at their default values.

The middle and right views in Figure 2 show the effect of increasing first `smin`, then `vmin`. In the right-hand view, noisy pixels have been largely eliminated, but the face region still produces a strong signal. Tracking is now quite good, and the ellipse is well positioned.

The Simple Camshift Wrapper

OpenCV includes source code for camshiftdemo, but it's not easy to adapt, since it combines user-input handlers and view toggling with the steps for face tracking.

If you're programming in C++, rather than in C, you could use the CvCamShiftTracker class, defined in cvaux.hpp. Again, however, this class is fairly complex, with many interfaces, and is only available to C++ programmers.

To make the Camshift tracker more accessible, I've written a wrapper for it in C with four main interfaces:

1) `createTracker()` pre-allocates internal data structures.

2) `releaseTracker()` releases these resources.

3) `startTracking()` initiates tracking from an image plus a rectangular region.

4) `track()` tracks the object in this region from frame to frame using Camshift.

There are two additional interfaces for setting the parameters vmin and smin:

1) `setVmin()`
2) `setSmin()`

The Camshift wrapper is online at **www.cognotics.com/opencv/downloads/camshift_wrapper/index.html**.

FIGURE 4. The helper functions `initAll()` and `exitProgram()` handle program initialization and cleanup.

Combining Face Detection and Tracking

In camshiftdemo, you needed to manually initialize tracking with the mouse. For a robotics application, it would be much nicer to initialize tracking automatically, using a face detection that the Haar detector returned. (See last month's article for details on implementing face detection.)

This section shows how to do that using the Camshift wrapper described above. The program described here detects a face in a live video stream, then tracks it with Camshift. The source for code for the complete program, called "Track Faces," is also available online at **www.cognotics.com/opencv/downloads/camshift_wrapper/index.html**.

The Main Program

Figure 3 shows the main program listing for detecting a face in a live video stream, then tracking it using the Camshift wrapper API. (This portion is in TrackFaces.c in the download.) There are three main program segments:

1) Detect a face.
2) Start the tracker.
3) Track the face.

1) *Detect a face.* Lines 15-27 implement a loop to examine video frames until a face is detected. The call to `captureVideoFrame()` invokes a helper method to bring in the next video frame and create a copy of it. (Recall from Part 1 of this series that it's never safe to modify the original video image!) The working copy is stored as `pVideoFrameCopy`, declared at line 6.

2) *Start the tracker.* When a face is detected, the code exits this loop (line 26) and starts the tracker (line 30), passing it the face rectangle from the Haar detector.

FIGURE 5. The helper function `captureVideoFrame()`. At line 11, the call to `cvFlip()` flips the image upside down if the origin field is 0.

initAll()

```
 1 int initAll()
 2 {
 3   if( !initCapture() ) return 0;
 4   if( !initFaceDet(OPENCV_ROOT
 5      "/data/haarcascades/haarcascade_frontalface_default.xml"))
 6      return 0;
 7
 8   // Startup message tells user how to begin and how to exit
 9   printf( "\n*****************************************\n"
10           "To exit, click inside the video display,\n"
11           "then press the ESC key\n\n"
12           "Press <ENTER> to begin"
13           "\n*****************************************\n" );
14   fgetc(stdin);
15
16   // Create the display window
17   cvNamedWindow( DISPLAY_WINDOW, 1 );
18
19   // Initialize tracker
20   captureVideoFrame();
21   if( !createTracker(pVideoFrameCopy) ) return 0;
22
23   // Set Camshift parameters
24   setVmin(60);
25   setSmin(50);
26
27   return 1;
28 }
```

exitProgram()

```
 1 void exitProgram(int code)
 2 {
 3   // Release resources allocated in this file
 4   cvDestroyWindow( DISPLAY_WINDOW );
 5   cvReleaseImage( &pVideoFrameCopy );
 6
 7   // Release resources allocated in other project files
 8   closeCapture();
 9   closeFaceDet();
10   releaseTracker();
11
12   exit(code);
13 }
```

3) *Track the face.* Lines 33-48 contain the face-tracking loop. Each call to the wrapper's `track()` method (line 41) invokes Camshift to find the face location in the current video frame. The Camshift result is returned as an OpenCV datatype called `CvBox2D`. This datatype represents a rectangle with a rotation angle. The call to `cvEllipseBox()` at lines 44-45 draws the ellipse defined by this box.

Helper Functions

In addition to the main program,

captureVideoFrame()

```
 1 void captureVideoFrame()
 2 {
 3   // Capture the next frame
 4   IplImage  * pVideoFrame = nextVideoFrame();
 5   if( !pVideoFrame ) exitProgram(-1);
 6
 7   // Copy it to the display image, inverting it if needed
 8   if( !pVideoFrameCopy )
 9     pVideoFrameCopy = cvCreateImage(cvGetSize(pVideoFrame),8,3);
10   cvCopy( pVideoFrame, pVideoFrameCopy, 0 );
11   if( 0==pVideoFrameCopy->origin ) cvFlip(pVideoFrameCopy,0,0);
12 }
```

References and Resources

- *OpenCV on Sourceforge* **http://sourceforge.net/projects/opencvlibrary**
- *Official OpenCV usergroup* **http://tech.groups.yahoo.com/group/OpenCV**
- G.R. Bradski, "Computer video face tracking for use in a perceptual user interface," *Intel Technology Journal*, Q2 1998.
- D. Comaniciu and P. Meer, "Robust Analysis of Feature Spaces: Color Image Segmentation," *CVPR*, 1997.
- *The Simple Camshift Wrapper* **www.cognotics.com/opencv/downloads/camshift_wrapper/index.html**
- Source code in this article can be downloaded from: **www.cognotics.com/opencv/servo**

TrackFaces.c also contains helper functions for initialization and cleanup — `initAll()` and `exitProgram()`. These are shown in Figure 4.

At line 21 in `initAll()`, the call to the Camshift wrapper's `createTracker()` function pre-allocates the wrapper's internal data structures. It's not necessary to pre-allocate the tracking data, but doing so speeds the transition from face detection to tracking. The next two statements (lines 24-25) set the parameters `smin` and `vmin`. The best values to use for these depends on your setup, so it's a good idea to select them ahead of time using the camshift-demo program, as described above.

Figure 5 shows the listing for `captureVideoFrame()`. At line 11, a call to `cvFlip()` flips the image upside down if the `origin` field is 0. The reason for doing this is that some web-cam drivers — especially on Windows — deliver image pixels starting at the bottom, rather than at the top, of the image. The `origin` field indicates which row order the IplImage uses. Some OpenCV functions will only work correctly when these images are inverted.

Finally, Figure 6 contains the `detectFace()` function. Although this code should be familiar from last month's article, one point worth noting is that the `min_neighbors` parameter should be set high enough that false face detections are unlikely. (Otherwise, your robot might start tracking the refrigerator magnets!) At line 10, I've set it to 6, which is more restrictive than the default value of 3.

detectFace()

```
1  CvRect * detectFace(IplImage * pImg)
2  {
3     CvRect* r = 0;
4
5     // detect faces in image
6     int minFaceSize = pImg->width / 5;
7     pFaceRectSeq = cvHaarDetectObjects
8        (pImg, pCascade, pStorage,
9        1.1,
10       6,
11       CV_HAAR_DO_CANNY_PRUNING,
12       cvSize(minFaceSize, minFaceSize));
13
14    // if one or more faces are detected, return the first one
15    if( pFaceRectSeq && pFaceRectSeq->total )
16       r = (CvRect*)cvGetSeqElem(pFaceRectSeq, 0);
17
18    return r;
19 }
```

FIGURE 6. The `detectFace()` function. The min_neighbors parameter is set to 6 to reduce the chance of a false detection.

Coming Up

So far, the faces we've been finding and following have been anonymous. The robot can tell there's a face present, and can follow it, but has no way of knowing whose face it is. The process of linking faces to names is called face recognition. OpenCV contains a complete implementation of a face-recognition method called eigenface.

The remaining two articles in this series will explain how to use OpenCV's eigenface implementation for face recognition. In the first of these, I'll explain how the algorithm works and give you code to create a database of people your robot "knows." The article following that takes you through the steps for recognition from live video, and gives you tips to help you get the most out of eigenface.

Be seeing you! SV