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.

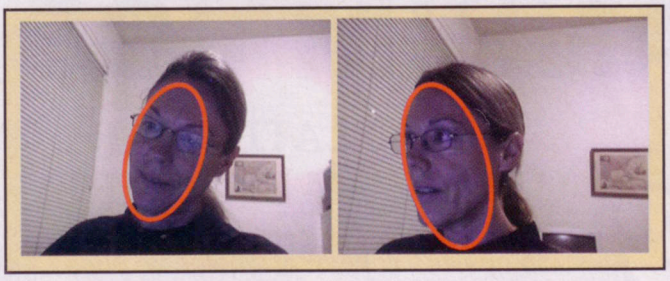

FIGURE 1. OpenCV's face tracker in action. It's able to follow a face as it tilts to one side and during a turn to profile.

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.

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.)

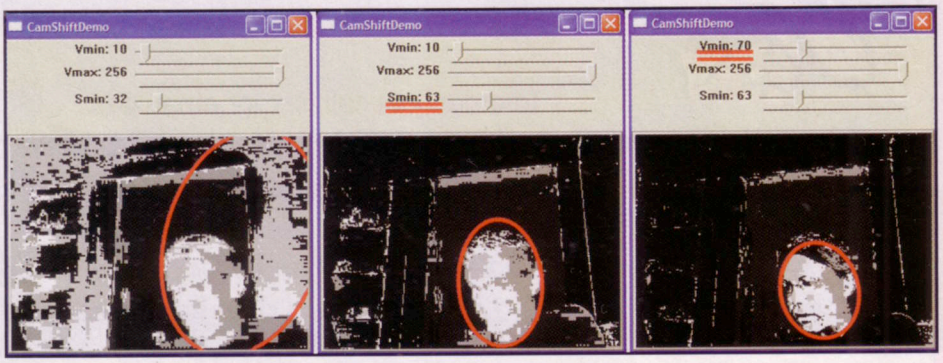

FIGURE 2. To tune the Camshift parameters `smin` and `vmin`, run the camshiftdemo program in the samples directory. These parameters are easier to set if you toggle to the backprojection view by clicking in the view window, then typing *b*.

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.

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()`.

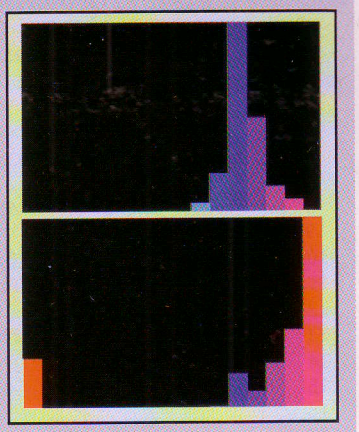

FIGURE A. Two examples of the color histogram that Camshift uses to represent a face.

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.

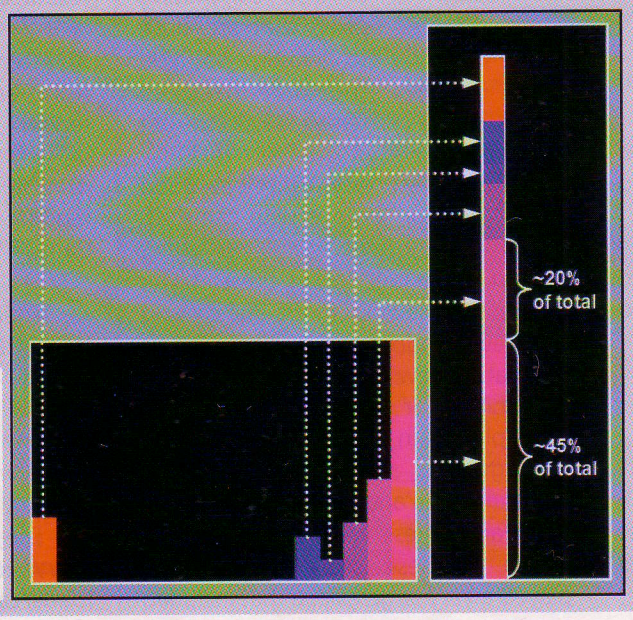

FIGURE B. To see what "face probability" means, imagine stacking the bars in a histogram one atop the other. The probability associated with each color is the percent that color bar contributes to the total height of this stack.

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.

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 webcam 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