OPEN SOURCE

Tweet Share Permalink

The OpenCV Library

By Gary Bradski, November 01, 2000

Post a Comment

OpenCV is an open-source, computer-vision library for extracting and processing meaningful data from images.

Gary is a researcher in Intel's Microprocessor Research Laboratory. He can be reached at gary.bradski@intel.com.

OpenCV is an open-source, computer-vision library for extracting and processing meaningful data from images. That meaningful data might include finding all or parts of objects, recognizing all or parts of objects, tracking the movement of (parts of) objects in 2D or 3D between successive images, determining the 2D or 3D shape of objects from one or more images, and associating image data with a categorical meaning, such as mapping a handwave to the category "goodbye." Developed by an Intel research group (of which I'm a member), OpenCV is freely available at http://www.intel .com/research/mrl/research/cvlib/. In this article, I'll provide an overview of OpenCV and present an example of gesture recognition for conducting music using the library.

OpenCV Overview

All in all, OpenCV contains more than 500 functions. While the API of the library is C/C++, the bulk of the library is made up of C functions and the objects themselves are mostly self contained. Objects and functions are *Extern*ed as C to avoid name mangling, allowing OpenCV to be used with Visual Basic and Java. Object-oriented design is mostly intended to take place at a COM or CORBA level for use in applications.

OpenCV is aimed at making computer vision accessible to programmers and users in the area of real-time human-computer interaction and mobile robotics. Thus, the library comes with source code and hand-tuned assembly language binaries optimized for Intel processors, so that users can both learn from the library and make use of its performance. Figure 1 is an example of OpenCV-based gesture recognition used to control music. When you build and run an app using OpenCV, a built-in DLL switcher is called at run time to automatically detect the processor type and load the appropriate optimized DLL for that processor. If the processor type cannot be determined (or if the appropriate DLL is not available), an optimized C code DLL is used. Included in the OpenCV download is the optimized Intel Image Processing Library (IPL) on which OpenCV partially depends. Although it is included with OpenCV, you can also get IPL and other libraries for signal processing, matrix math, JPEG, and pattern recognition at http://developer.intel.com/vtune/ perflibst/.

OpenCV ships with an HTML overview and a detailed manual in PDF format. Added to this are many source-code examples, detailed papers, and tutorials on different topics. The source-code examples currently include camera calibration, face tracking, Kalman filter, condensation filter, face recognition, optical flow, and morphing an image to produce intermediate views between two real camera views. An interpretive C prototyping environment for OpenCV and IPL is also available for download. OpenCV currently is supported to run on Windows 98/NT/2000. The source code should build and run on Linux or other operating systems. Official Linux support is currently underway.

Content

Open Source Recent Articles

Dr. Dobb's Archive
Jolt Awards 2015: Coding Tools
The Long Death of Project Hosting Sites
Building Portable Games in C++
Program Configuration in Python

Most Popular

Stories Blogs

Why Build Your Java Projects with Gradle Rather than Ant or Maven?
Hadoop: Writing and Running Your First Project
A Simple and Efficient FFT Implementation in C++: Part I
Unit Testing with Python
Jolt Awards 2015: Coding Tools

This month's Dr. Dobb's Journal

Dr. Dobb's Digital Digest - October 2014
This month, Dr. Dobb's Journal is devoted to mobile programming. We introduce you to Apple's new Swift programming language, discuss the perils of being the third-most-popular mobile platform, revisit SQLite on Android , **and much more!**

Download the latest issue today. >>

Upcoming Events

Live Events WebCasts

No records found

Featured Reports

What's this?

State of ITSM in Manufacturing
The Rise of the No-Code Economy
2021 Digital Transformation Report
CVE Weaponization Report
Why You're Wrong About Operationalizing AI

More >>

Featured Whitepapers

What's this?

The Burnout Breach: How employee burnout is emerging as the next frontier in cybersecurity
Selling Breaches: The Transfer of Enterprise Network Access on Criminal Forums
Defending Corporate Executives and VIPs from Cyberattacks
The Impact of XDR in the Modern SOC
Supply Chain Cyber Risk Management Whitepaper

More >>

Most Recent Premium Content

ZESTY.AI EX1019
U.S. Patent No. 10,529,029

OpenCV addresses the areas of object/human/face segmentation, detection, recognition, and tracking, as well as camera calibration, stereovision, and 2D/3D shape reconstruction. A full matrix algebra package is also included in the library to support algorithms in these areas.

Also included in the library are routines in:

- Geometric Methods: contours, space tessellation, and triangulation.
- Image measures: image statistics, spatial moments, and contour moments.
- Utilities: image pyramids, data structures, linked lists, image management, mathematical functions, fast pixel access, line and conic section drawing, and text display.
- Segmentation: image morphology, thresholding, color and texture pyramids, background subtraction, histogram backprojection, HMM viterbi, K-means, and normalized cut.
- Feature Detection: Hough transform, Canny edge detector, corner detection, subpixel accurate line and corner location, contours, image derivatives to the third order.
- Recognition: histogram matching, template matching, Mahalanobis distance, HMM, embedded HMM, shape descriptors, Eigen objects, LDA, 3D gesture.
- Tracking: mean-shift, CAMSHIFT, optical flow, affine flow, motion templates, energy snakes, Kalman and Condensation filters.
- Camera: calibration, view morphing, 8-point algorithm, correspondence, and stereo support.
- Shape: 2D and 3D line and ellipse fitting, and shape toolbox.

In addition, most textbook image processing routines are covered by the OpenCV download via a separate optimized IPL that is fully compatible with OpenCV.

A Conducting Music Example

To illustrate how to use OpenCV, I'll present an application that detects waving and clapping motions to conduct music. Figure 2 is a flowchart of this algorithm. When a new video image comes in, the foreground is separated from the background using background subtraction; the centroid of this foreground object is identified and stored. The foreground object is layered (with a value equal to the current system time stamp) onto the "timed Motion History Image" (tMHI). Previous layers that are too old in the tMHI are deleted. Image gradients (directions of steepest accent) of the tMHI are calculated, the gradient directions indicate movement patterns of the foreground object. The gradients are masked into several regions relative to the centroid of the foreground object. In this example, a person is the foreground object and I mask motions to the left center, left top, right top, and right center. The motion orientations in these regions are then used to infer waving and clapping motions that are in turn used to control the tempo and spatial orientation of synthesized vocal harmonies.

For more details on the algorithm and code just described, go to http://www.eecs .lehigh.edu/FRAME/Davis/DavisBradski.htm. Here, I will only discuss background segmentation, building the tMHI, and turning it into a direction of motion representation.

Simple Foreground Object Segmentation

For this discussion, assume the camera is stationary. Although there are many more sophisticated techniques of segmenting a foreground object (person) from a learned background, in this example I label as potential foreground those pixels that are a set number of standard deviations from the mean color (Red, Green, Blue) background. In order to do this, I need to first learn the background model. The learned background model will consist of the means and standard deviations of each pixel over many frames when no foreground object is present. Listing One shows the routine that does this. The pixel values and squared values of pixels are summed into floating-point images and these images are divided by the number of video frames (here 45 frames or ~1.5 seconds at 30 frames/sec.) after collection is finished. This yields the mean (*Imean[x,y]*) and standard deviation (*IstdFP[x,y]*) of each pixel. Since *IstdFP* will be used as the threshold, I make sure that no standard deviation value is too small. Finally, *backThresh* standard deviations (that is, *backThresh*IstdFP[x,y]*) is the threshold

Digital Issues

2014
Dr. Dobb's Journal
November - **Mobile Development**
August - **Web Development**
May - **Testing**
February - **Languages**

Dr. Dobb's Tech Digest
DevOps
Open Source
Windows and .NET programming
The Design of Messaging Middleware and 10 Tips from Tech Writers
Parallel Array Operations in Java 8 and Android on x86: Java Native Interface and the Android Native Development Kit

2013
January - **Mobile Development**
February - **Parallel Programming**
March - **Windows Programming**
April - **Programming Languages**
May - **Web Development**
June - **Database Development**
July - **Testing**
August - **Debugging and Defect Management**
September - **Version Control**
October - **DevOps**
November- **Really Big Data**
December - **Design**

2012
January - **C & C++**
February - **Parallel Programming**
March - **Microsoft Technologies**
April - **Mobile Development**
May - **Database Programming**
June - **Web Development**
July - **Security**
August - **ALM & Development Tools**
September - **Cloud & Web Development**
October - **JVM Languages**
November - **Testing**
December - **DevOps**

difference from *Imean[x,y]* at which a future pixel at *x,y* will be declared to be a foreground pixel.

After the mean and the standard deviation of the color background are calculated, the code then creates a binary mask of all possible foreground regions. Foreground regions are those pixels that deviate more than a given number of standard deviations from the mean values of red, green, or blue pixels. Image dilation (expanding the boundary of foreground objects) is used to help close holes in the foreground regions caused by image noise as in Listing Two, which makes use of image-processing library routines. Because the DLL switcher automatically loads the optimized assembly DLL or the optimized C version of the DLL if the former can't be found, you can easily compare the speed-up between optimized C and optimized assembly (which takes advantage of the SIMD instruction set). This foreground hypothesis routine is 3.71 times faster using optimized assembly than when using optimized C on a Pentium III.

The next routine in the code searches through all the potential foreground regions to find the biggest region that is greater than a minimum size threshold. The search is performed with successive floodfills; see Listing Three. If a foreground region is too small, it is zeroed out; if a foreground region larger than the current foreground region is found, the smaller region is zeroed out and the larger region is retained.

Updating and Using the tMHI

After the video image has been processed for a foreground object, the code updates the tMHI. The current system timestamp is obtained in the format *seconds.milliseconds*. The current foreground object, if any, is overwritten on top of the tMHI with values equal to the system timestamp. The tMHI is then searched to delete any time values older than a given threshold. The process is depicted in Figure 3.

In the OpenCV library, updating the tMHI image is wrapped into one function call, *cvUpdateMHIByTime (...).* The image gradients are calculated from the tMHI via the call *cvCalcMotionGradient (...)*. This yields directions of motion encoded implicitly by the tMHI. Finally, the code computes the global direction of motion within that region via the call *cvCalcGlobalOrientation (...).* These calls are shown in Listing Four. Figure 4 depicts the process of going from the tMHI, to a gradient representation to the global motion within that region.

Table 1 lists code timing for some of the OpenCV functions involved in this process. The timings are in the form of clock cycles per pixel, and the speed-up factor in going from optimized C code to optimized assembly is shown in this column. The total algorithm, including masking and calculating motion in four regions around the centroid of the person (see Figure 1) and mapping these detected motions into music conducting controls, takes 140 cycles per pixel. Thus, if I work with 160 X 120 video images at 30-Hz on a 500-MHz Pentium III, this algorithm would use about 1/6 of the CPU leaving time to do things with the recognized gestures like synthesize music.

Conclusion

As computers increase in performance and cameras decrease in cost, tools such as OpenCV will help set the infrastructure in place for new ways of interacting with computing devices, especially as computers move into set-top boxes, handhelds, projection screens, and wearable computers.

Further Reading

Bradski, G., B-L Yeo, and M. Yeung. "Gesture for video content navigation." *SPIE '99*, 3656-24 S6, 1999.

Bradski, G. "Computer Vision Face Tracking For Use in a Perceptual User Interface." *Intel Technology Journal*. http://developer.intel.com/technology/itj/q21998/articles/art_2.htm. Q2, 1998.

Davis, J. and Gary Bradski. "Real-time Motion Template Gradients using Intel CVLib." *IEEE ICCV '99 Frame-Rate Workshop*, 1999. http://www.eecs.lehigh.edu/FRAME/Davis/DavisBradski.htm.

Grzeszczuk, R., Bradski, G., Chu, M. and Bouguet J-Y. "Stereo Based Gesture Recognition Invariant to 3D Pose and Lighting." *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2000.

Nefian, Ara V. and Monson H. Hayes III. "Face Recognition using an embedded HMM." *IEEE International Conference Audio Video Biometric-based Person Authentication*, March 1999.

DDJ

Listing One

```
//Collect background mean and std  of each pixel for background differencing
void captureBackground(IplImage *IvidIm, IplImage *IstdFP, IplImage *Iu, IplImage *Istd)
{
        int height = IvidIm->height;
        int width = IvidIm->width;
        int i;
        //Create background mask (find mean and variance of color background):
        IplImage *Imean = iplCreateImageHeader(3, 0, IPL_DEPTH_32F, "RGB", "BGR",
            IPL_DATA_ORDER_PIXEL, IPL_ORIGIN_TL, IPL_ALIGN_QWORD,
             width, height, NULL, NULL, NULL, NULL);
        iplAllocateImageFP(Imean, 1,0.0);
        int len = width*height*3;
        for(i = 0; i<45; i++) //take statistics over 45 frames (~1.5 secs)
        {
                grabIm(IvidIm);                 //Get an image into IvidIm
                cvAcc(IvidIm, Imean);           //Accumulate it into Imean
                cvSquareAcc(IvidIm, IstdFP);    //Accumulate squared image into IstdFP
        }
        //find mean and vars
        iplMultiplySFP(Imean, Imean, (float)1.0/i);     //meanI
        iplMultiplySFP(IstdFP,  IstdFP,  (float)1.0/i); //meanI^2
        IplImage* ImeanSqr = iplCloneImage(Imean);
        iplSquare(ImeanSqr, ImeanSqr);
        iplSubtract(IstdFP, ImeanSqr, IstdFP);  //Ivar = meanI^2 - (meanI)^2
    iplDeallocate(ImeanSqr, IPL_IMAGE_ALL);
        cvbSqrt((const float*)IstdFP->imageData,
                (float*)IstdFP->imageData, len);        //IstdFP = sqrt(Ivar)
        //since we use Istd as a threshold, enforce that no threshold is too small:
        float *pIstdFP = (float *)IstdFP->imageData;
        for(i=0; i<len; i++)
        {
                if(*pIstdFP < 0.3)
                    *pIstdFP = 0.3;
                pIstdFP++;
        }
        iplMultiplySFP(IstdFP,  IstdFP,  backThresh);   //meanI^2
        //convert to 8u images
        convert32Fto8U(Imean,Iu);
        convert32Fto8U(IstdFP,Istd);
    iplDeallocate(Imean, IPL_IMAGE_ALL);
}
```

Back to Article

Listing Two

```
//Extract the foreground and fill in holes
//Ii            Video input image  BGR
//Im            Mean image of background  BGR
//Is            Standard deviation of background from mean BGR
//Io            Output image            -- Grayscale
//Iot           Temporary output image  -- Grayscale
//It1,It2       Temporary images of same size, depth and number of channels BGR
//numIterations Number of dialations to preform on foreground
void backsubCVL(IplImage *Ii, IplImage *Im, IplImage *Is, IplImage *Io,
                        IplImage *Iot, IplImage *It1, IplImage *It2, int numIterations)
{
        //Get |Ii-Im|
        iplSubtract(Ii,Im,It1);
        iplSubtract(Im,Ii,It2);
        iplAdd(It1,It2,It1);
        //Get Raw foreground = |Ii-Im| > Is ? 255 : 0;
        iplSubtract(It1,Is,It1);
        iplThreshold(It1, It1, 1);
        iplColorToGray(It1, Io);
        iplThreshold(Io,Iot,1);
        //Fill up holes in the foreground
        iplDilate(Iot,Io, numIterations);
}
```

Back to Article

Listing Three

```
// Get the largest region in the silhouette image
// Isil          Image with "foreground" regions in it 8uC1
// width,height Of image
// MIN_REG...    Minimum size acceptable to declare a foreground object
// cc2           Return: The bounding box and area of found region
//
//Returns       Area of largest region found, else 0 = nothing found greater than
int growRegionCVL(IplImage *Isil, int width, int height,
                int MIN_REGION_SIZE, CvConnectedComp &cc2)
{
        cc2.area = 0.0;
        int biggestNum=0,biggestLocX=0,biggestLocY=0;
        int haveOne=0;
        int x = 0,y = 0;

        CvPixelPosition8u sil;
        CvSize silSize;
        silSize.width = width;
        silSize.height = height;
        unsigned char * pI = (unsigned char *)Isil->imageData;
        CV_INIT_PIXEL_POS(sil, pI, Isil->widthStep, silSize, 0, 0,IPL_ORIGIN_TL);
        int stride = Isil->widthStep;
        CvPoint xy;
        for(y=0; y<height; y++)
        {
                for(x=0; x<width; x++)
                {
                        // check if used yet
                    if(*(sil.currline + sil.x) == 255)
                    {
                            xy.x = x;
                            xy.y = y;
                            cvFloodFill ( Isil, xy, 100, 0, 0, &cc2);
                            // if size is too small remove that region
                                    // Also, keep only the biggest region!!!
                            if( ((int)(cc2.area)<MIN_REGION_SIZE) ||
                                    ((int)(cc2.area)<biggestNum) )
                                    {
                                            // remove it
                                    cvFloodFill ( Isil, xy, 0, 0, 0, &cc2);
                                    }
                                    else // for keeping just
                                    {
                                    // remove previous max
                                    if(haveOne)
                                    {
                                            xy.x = biggestLocX;
                                            xy.y = biggestLocY;
                                            cvFloodFill ( Isil, xy, 0,
                                    }
                                    else
                                            haveOne=1;
                                    biggestNum=(int)(cc2.area);
                                    biggestLocX=x;
                                    biggestLocY=y;
                            }
                    }//end if potential point found
                    CV_MOVE_RIGHT_WRAP(sil, 1);
                    }//end for x
                CV_MOVE_DOWN(sil, 1);
        }//end for y
        if(haveOne)
        {
                xy.x = biggestLocX;
                xy.y = biggestLocY;
                cvFloodFill ( Isil, xy, 255, 1, 1, &cc2);
                return(biggestNum);
        }
        else
                return(0);
}
```

Back to Article

Listing Four

```
//UPDATE THE MOTION HISTORY IMAGE
cvUpdateMHIByTime(IsilIm, ImhiIm, timestamp, MHI_DURATION);

//CREATE MOTION GRADIENT ORIENATIONS FROM THE tMHI
cvCalcMotionGradient(ImhiIm, Imask, Iorient, 3, MAX_TIME_DELTA,MIN_TIME_DELTA );

//CALCULATE THE MOTION ORIENTATION
globalDir = cvCalcGlobalOrientation( Iorient, Imask, ImhiIm, timestamp,
globalDir = 360.0 - globalDir;      //Circularly rotate angle since TL origin rather                    //than BL
```

Back to Article

1 2 3 4 5 6 Next

Related Reading

- News
- Commentary

- **Jelastic Docker Integration For Orchestrated Delivery**
- **Restlet Completes "Complete" API Platform**
- **Mac OS Installer Platform From installCore**
- **New Relic Continues Developer Data Apps Push**
  More News»

- Slideshow
- Video

- **Jolt Awards 2014: The Best Testing Tools**
- **The Most Underused Compiler Switches in Visual C++**
- **Jolt Awards: The Best Testing Tools**
- **Developer Reading List**
  More Slideshows»

- Most Popular

- **RESTful Web Services: A Tutorial**
- **Lambda Expressions in Java 8**
- **Developer Reading List: The Must-Have Books for JavaScript**
- **Why Build Your Java Projects with Gradle Rather than Ant or Maven?**
  More Popular»

More Insights
White Papers

- Build a Case for a Password Manager
- 2021 Gartner Market Guide for Managed Detection and Response Report

More >>

Reports

- Forrester Total Economic Impact Study: Team Cymru Pure Signal Recon
- The 2021 Security Outcomes Study

More >>

Webcasts

- Does Your Network - NOT WORK?
- [FREE Virtual Event] IT Automation in 2023: Enabling Innovation and Efficiency

More >>

INFO-LINK

Login or Register to Comment

Discover More From Informa Tech

Working With Us

InformationWeek
Dark Reading
Network Computing

Interop
Data Center Knowledge
IT Pro Today

Contact Us
About Us
Advertise
Reprints

Follow Dr. Dobb's On Social

Copyright © 2025 Informa PLC. Informa PLC is registered in England and Wales with company number 8860726 whose registered and head office is 5 Howick Place, London, SW1P 1WG.