OpenCV Tutorial 1 - Chapters 1 and 2

Author: Noah Kuntz (2008)
Contact: nk752@drexel.edu

Keywords: OpenCV, computer vision, image manipulaton, gaussian blur, avi

My Vision Tutorials Index

This tutorial assumes the reader:
(1) Has a basic knowledge of Visual C++
(2) Has some familiarity with computer vision concepts

The rest of the tutorial is presented as follows:

Important Note!

More information on the topics of these tutorials can be found in this book: Learning OpenCV: Computer Vision with the OpenCV Library

Step 1: Installing OpenCV

Install MSVC++ on windows and GCC or GLIBC on Linux.

Download OpenCV from sourceforge

Use opencv-win or opencv-lin depending on your platform of choice.

Optionally you may purchase Intel Integrated Performance Primitives (IPP) which optimizes the performance of OpenCV. Intel IPP

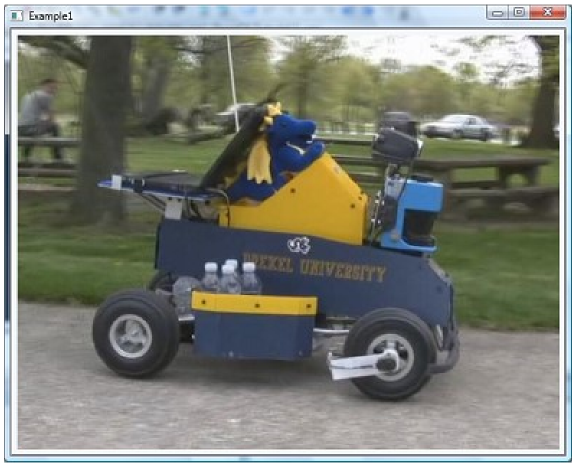
Optionally update OpenCV using the "Concurrent Versions System," with software like Tortoise CVS.

Now you must create a C++ project and link the correct files. These headers should be linked:

```
#include "C:\Program Files\OpenCV\cv\include\cv.h"
#include "C:\Program Files\OpenCV\ml\include\ml.h"
#include "C:\Program Files\OpenCV\cxcore\include\cxcore.h"
#include "C:\Program Files\OpenCV\cxcore\include\cxtypes.h"
#include "C:\Program Files\OpenCV\otherlibs\highgui\highgui.h"
```

Link to cv.lib, ml.lib, cxcore.lib, and highgui.lib Make sure to change the environment variable to PATH=C:\Program Files\OpenCV\bin

Step 2: Displaying Images and Video



An image displayed with OpenCV

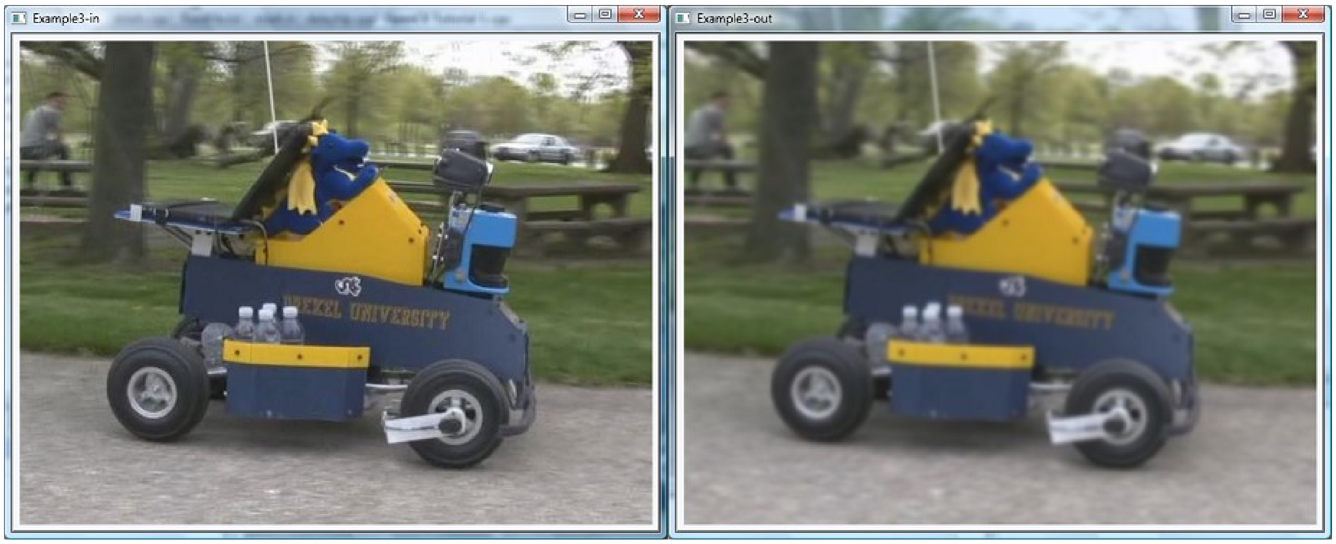
The first thing we can do with OpenCV is simply access an image. This code will display a the image "MGC.jpg" in a window. *cvLoadImage* loads the image into an IplImage* variable. *cvNamedWindow* creates a window and *cvShowImage* paints the image on the window. Then *cvWaitKey(0)* waits for any key to be pressed at which point the memory is released and the window is cleared with *cvReleaseImage* and *cvDestroyWindow*.

```
int _tmain(int argc, _TCHAR* argv[])
{
        IplImage* img = cvLoadImage( "MGC.jpg" );
        cvNamedWindow( "Example1", CV_WINDOW_AUTOSIZE );
        cvShowImage("Example1", img);
        cvWaitKey(0);
        cvReleaseImage( &img );
        cvDestroyWindow( "Example1" );
        return 0;
}
```

The next step is to display a video. Now we have to create a *CvCapture** object with *cvCreateFileCapture*. We loop through the images using *cvQueryFrame* to get each frame and *cvWaitKey(33)* to wait 33ms between each frame. Then if the user presses Esc the loop breaks and the capture is released and the window closed. It should be noted that in order to capture from a camera instead of a video file, all that is needed is to replace *cvCreateFileCapture* with *cvCreateCameraCapture(0)* which will pick the first available camera interface. Here is the code for playing a video:

```
int _tmain(int argc, _TCHAR* argv[])
{
        cvNamedWindow( "Example2", CV_WINDOW_AUTOSIZE );
        CvCapture* capture = cvCreateFileCapture( "MGC_RC_ATV.avi" );
        IplImage* frame;
        while(1) {
                frame = cvQueryFrame( capture );
                if( !frame ) break;
                cvShowImage( "Example2", frame );
                char c = cvWaitKey(33);
                if( c == 27 ) break;
        }
        cvReleaseCapture( &capture );
        cvDestroyWindow( "Example2" );
        return 0;
}
```

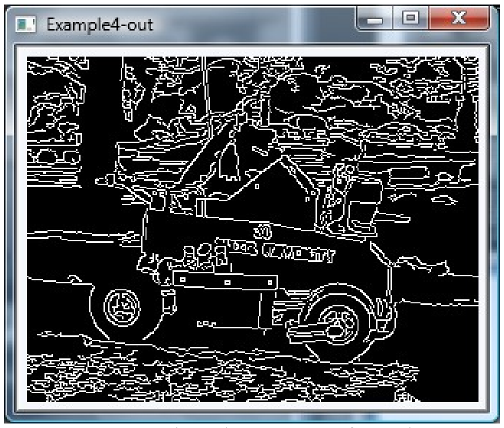
Step 3: Simple Transformations



Gaussian blur transformation

Next we would like to be able to do some actual processing on an image. One simple transformation is a gaussian blur. First we load an image as before, and make two windows. A new function in this example is creating an empty image to hold the output, with *cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3)* which is saying make a new image the same size as "img", with 3 channels of 8 bits each. Then the gaussian blur is accomplished with *cvSmooth(img, out, CV_GAUSSIAN, 11, 11)* which puts the result in "out" and uses a window of 11 x 11 for the blur. Here is the code:

```
int _tmain(int argc, _TCHAR* argv[])
{
        IplImage* img = cvLoadImage( "MGC.jpg" );
        cvNamedWindow( "Example3-in" );
        cvNamedWindow( "Example3-out" );

        // Show the original image
        cvShowImage("Example3-in", img);

        // Create an image for the output
        IplImage* out = cvCreateImage( cvGetSize(img), IPL_DEPTH_8U, 3 );

        // Perform a Gaussian blur
        cvSmooth( img, out, CV_GAUSSIAN, 11, 11 );

        // Show the processed image
        cvShowImage("Example3-out", out);

        cvWaitKey(0);
        cvReleaseImage( &img );
        cvReleaseImage( &out );
        cvDestroyWindow( "Example3-in" );
        cvDestroyWindow( "Example3-out" );
        return 0;
}
```



Downsample and Canny transformation

Here is an example of more transformations, doing a pyramidal downsample of an image, and performing Canny edge detection. There are a few new things in this example. By adding an input of 0 to cvLoadImage the image is forced to be grayscale. This is required for Canny edge detection. Also we use properties of the image loaded, for example "img->width." Using "->" you can see all the properties for the image that are available. And then the image is processed using *cvPyrDown(img, out)* for the downsample, and then *cvCanny(out, out, 10, 100, 3)* for the edge detection. For the canny function the syntax is "input," "output," "lower threshold," "upper threshold," and "aperature."

```
int _tmain(int argc, _TCHAR* argv[])
{
        IplImage* img = cvLoadImage( "MGC.jpg", 0);
        cvNamedWindow( "Example4-in" );
        cvNamedWindow( "Example4-out" );

        // Show the original image
        cvShowImage("Example4-in", img);

        // Make sure image is divisible by 2
        assert( img->width%2 == 0 && img->height%2 == 0);

        // Create an image for the output
        IplImage* out = cvCreateImage( cvSize(img->width/2,img->height/2), img->depth, img->nChannels );

        // Reduce the image by 2
        cvPyrDown( img, out );

        // Perform canny edge detection
        cvCanny( out, out, 10, 100, 3 );

        // Show the processed image
        cvShowImage("Example4-out", out);

        cvWaitKey(0);
        cvReleaseImage( &img );
        cvReleaseImage( &out );
        cvDestroyWindow( "Example4-in" );
        cvDestroyWindow( "Example4-out" );

        return 0;
}
```

Final Words

This tutorial's objective was to show how to do basic image loading and processing in OpenCV. Later tutorials in this series will expand on the functionality shown here.

Click here to email me.
Click here to return to my Tutorials page.