OpenCV Tutorial 4 - Chapter 5

Author: Noah Kuntz (2009)
Contact: nk752@drexel.edu

Keywords: OpenCV, computer vision, filter, image processing, blur, truncate, threshold, gaussian, erode, dilate

My Vision Tutorials Index

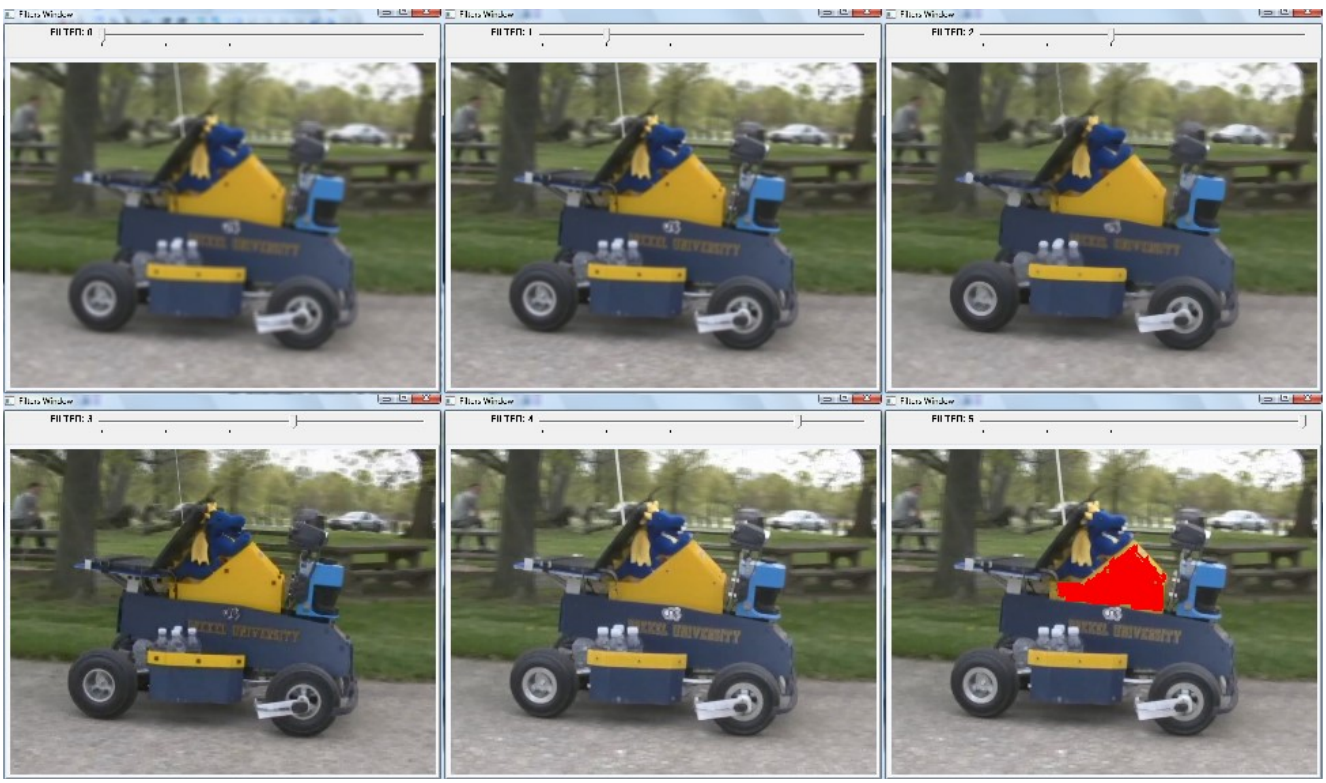
This tutorial assumes the reader:
(1) Has a basic knowledge of Visual C++
(2) Has some familiarity with computer vision concepts
(3) Has read the previous tutorials in this series

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: Filter Examples



Six different filters applied to an image

This chapter presents the use of several basic image processing filters. The best way to understand these filters in a general sense is to see them in action. For more technical details about their function please see the text. This example will demonstrate blur, gaussian, and median smoothing using *cvSmooth*. Smoothing operations are often an important precursor to other processing. Then *cvErode*, *cvDilate* and *cvFloodFill* are demonstrated. All of these can be helpful when segmenting an image, amoung other things. Here is the code:

```
int g_switch_value = 0;
int filterInt = 0;
int lastfilterInt = -1;
```

```
void switch_callback( int position ){
        filterInt = position;
}

int _tmain(int argc, _TCHAR* argv[])
{
        const char* name = "Filters Window";
        IplImage* img = cvLoadImage( "MGC.jpg" );
        IplImage* out = cvCreateImage( cvGetSize(img), IPL_DEPTH_8U, 3 );

        cvNamedWindow( name, 1 );
        cvShowImage(name, out);

        // Other Variables
        CvPoint seed_point = cvPoint(305,195);
        CvScalar color = CV_RGB(250,0,0);

        // Create trackbar
        cvCreateTrackbar( "FILTER", name, &g_switch_value, 5, switch_callback );

        while( 1 ) {
                switch( filterInt ){
                        case 0:
                                cvSmooth( img, out, CV_BLUR, 7, 7 );
                                break;
                        case 1:
                                cvSmooth( img, out, CV_GAUSSIAN, 7, 7 );
                                break;
                        case 2:
                                cvSmooth( img, out, CV_MEDIAN, 7, 7 );
                                break;
                        case 3:
                                cvErode( img, out, NULL, 1);
                                break;
                        case 4:
                                cvDilate( img, out, NULL, 1);
                                break;
                        case 5:
                                cvFloodFill( out, seed_point, color, cvScalarAll(5.0), cvScalarAll(5.0), NULL, 4, NULL );
                                break;
                }
                if(filterInt != lastfilterInt){
                        cvShowImage(name, out);
                        lastfilterInt = filterInt;
                }
                if( cvWaitKey( 15 ) == 27 )
                        break;
        }

        cvReleaseImage( &img );
        cvReleaseImage( &out );
        cvDestroyWindow( name );

        return 0;
}
```

Step 2: Thresholding

Thresholded Image

Another basic filter is thresholding. This code shows how to do a truncation, where values over 100 are discarded. This is of course not the same as a binarization thresholding operation, where everything that is above one value is white and everything below is black. *cvThreshold* is able to take five

possible values: *CV_THRESH_BINARY, CV_THRESH_BINARY_INV, CV_THRESH_TRUNC, CV_THRESH_TOZERO, CV_THRESH_TOZERO_INV*. Binary and binary inverse should be self explanitory, and truncate is shown in this example. To zero is essentially the opposite of truncate (preserving everything below and discarding values above).

```
void sum_rgb( IplImage* src, IplImage* dst ){

        // Allocate image planes
        IplImage* r = cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
        IplImage* g = cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );
        IplImage* b = cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );

        // Split image onto the color planes
        cvSplit( src, r, g, b, NULL );

        IplImage* s = cvCreateImage( cvGetSize(src), IPL_DEPTH_8U, 1 );

        // Add equally weighted rgb values
        cvAddWeighted( r, 1./3., g, 1./3., 0.0, s );
        cvAddWeighted( s, 2./3., b, 1./3., 0.0, s );

        // Truncate values over 100
        cvThreshold( s, dst, 100, 100, CV_THRESH_TRUNC );

        cvReleaseImage( &r );
        cvReleaseImage( &g );
        cvReleaseImage( &b );
        cvReleaseImage( &s );
}

int _tmain(int argc, _TCHAR* argv[])
{
        const char* name = "Thresholding";
        cvNamedWindow( name, 1 );

        IplImage* src = cvLoadImage("MGC.jpg");
        IplImage* dst = cvCreateImage( cvGetSize(src), src->depth, 1 );
        sum_rgb( src, dst);

        cvShowImage( name, dst);

        while( 1 ){
                if( (cvWaitKey(10)&0x7f) == 27 )
                        break;
        }

        cvDestroyWindow( name );
        cvReleaseImage( &src );
        cvReleaseImage( &dst );

        return 0;
}
```

Final Words

This tutorial's objective was to show how to use some image filters functions. You should be able to extend the use of these functions for more elaborate filtering operations.

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