OpenCV Tutorial 5 - Chapter 6

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

Keywords: OpenCV, computer vision, image processing, edge detection, canny, transforms, affine

My Vision Tutorials Index

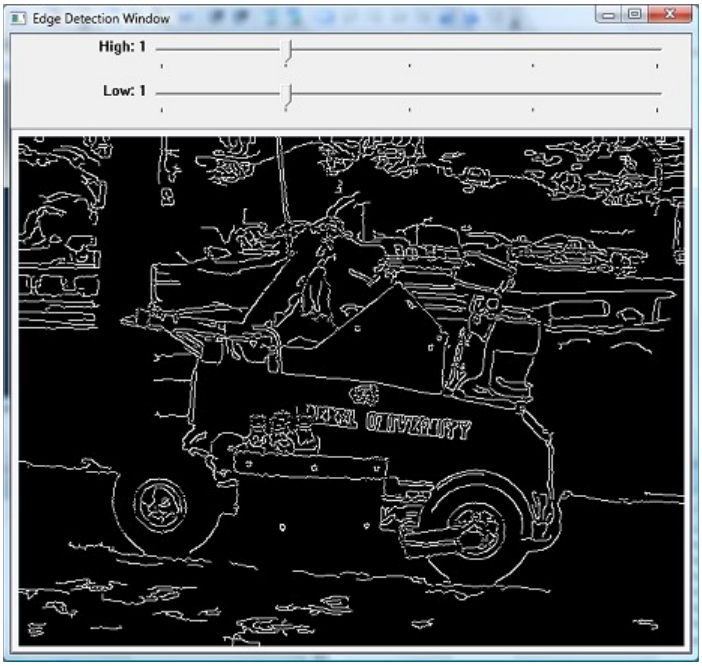
This tutorial assumes the reader:
(1) Has a basic knowledge of Visual C++
(2) Has some familiarity with computer vision concepts
(3) Has read the previous tutorials in this series

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: Edge Detection



Canny Edge Detection

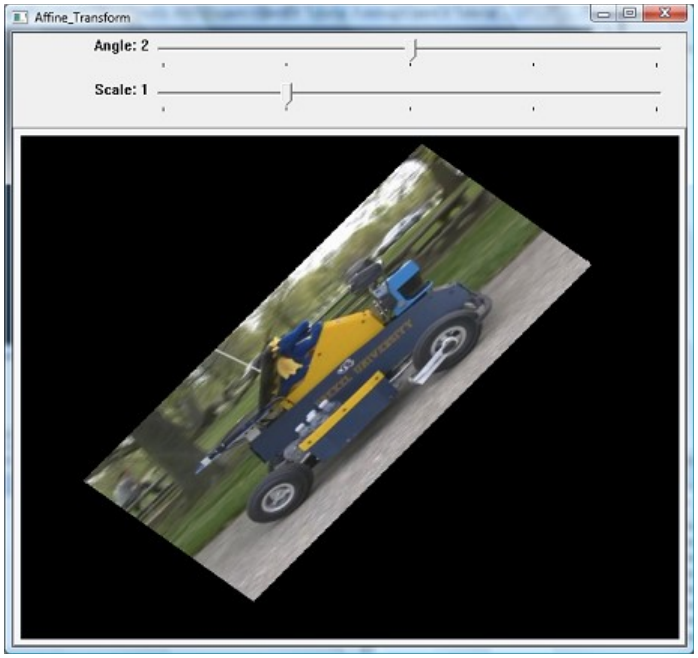
This chapter presents the use of several image transforms. The first transform I will cover is Canny Edge Detection. The book describes Sobel and Laplace operators for gradient detection, for more technical details about their function please see the text. The Canny Edge Detection algorithm takes the derivative of an image to find the gradients, then determines the direction of these gradients (vertical, horizontal, diagonal up, diagonal down). Then if the amplitude of a given gradient is high enough (the high threshold), the algorithm will trace along that gradient in its direction until the amplitude falls below the low threshold, or the gradient changes direction sharply. The algorithm will also suppress local non-maximums around the edges. *cvCanny* implements this algorithm. The inputs are the two images to be used, the thresholds, and the aperature size of the convolution kernal. For this example I created two sliders that adjust the

thresholds, try moving the sliders and see the effect of the thresholds on the resulting edges. This example also makes use of *cvCopyMakeBorder* to pad the borders of the image for convolution. This is not required but can improve results. Here is the code:

```
int high_switch_value = 0;
int highInt = 0;
int low_switch_value = 0;
int lowInt = 0;

void switch_callback_h( int position ){
        highInt = position;
}
void switch_callback_l( int position ){
        lowInt = position;
}

int _tmain(int argc, _TCHAR* argv[])
{
        const char* name = "Edge Detection Window";

        // Kernel size
        int N = 7;

        // Set up images
        IplImage* img = cvLoadImage( "MGC.jpg", 0 );
        IplImage* img_b = cvCreateImage( cvSize(img->width+N-1,img->height+N-1), img->depth, img->nChannels );
        IplImage* out = cvCreateImage( cvGetSize(img_b), IPL_DEPTH_8U, img_b->nChannels );

        // Add convolution boarders
        CvPoint offset = cvPoint((N-1)/2,(N-1)/2);
        cvCopyMakeBorder(img, img_b, offset, IPL_BORDER_REPLICATE, cvScalarAll(0));

        // Make window
        cvNamedWindow( name, 1 );

        // Edge Detection Variables
        int aperature_size = N;
        double lowThresh = 20;
        double highThresh = 40;

        // Create trackbars
        cvCreateTrackbar( "High", name, &high_switch_value, 4, switch_callback_h );
        cvCreateTrackbar( "Low", name, &low_switch_value, 4, switch_callback_l );

        while( 1 ) {
                switch( highInt ){
                        case 0:
                                highThresh = 200;
                                break;
                        case 1:
                                highThresh = 400;
                                break;
                        case 2:
                                highThresh = 600;
                                break;
                        case 3:
                                highThresh = 800;
                                break;
                        case 4:
                                highThresh = 1000;
                                break;
                }
                switch( lowInt ){
                        case 0:
                                lowThresh = 0;
                                break;
                        case 1:
                                lowThresh = 100;
                                break;
                        case 2:
                                lowThresh = 200;
                                break;
                        case 3:
                                lowThresh = 400;
                                break;
                        case 4:
                                lowThresh = 600;
                                break;
```

```
                }

                // Edge Detection
                cvCanny( img_b, out, lowThresh*N*N, highThresh*N*N, aperature_size );
                cvShowImage(name, out);

                if( cvWaitKey( 15 ) == 27 )
                        break;
        }

        // Release
        cvReleaseImage( &img );
        cvReleaseImage( &img_b );
        cvReleaseImage( &out );
        cvDestroyWindow( name );

        return 0;
}
```

Step 2: Affine Transforms



Affine Transform Example

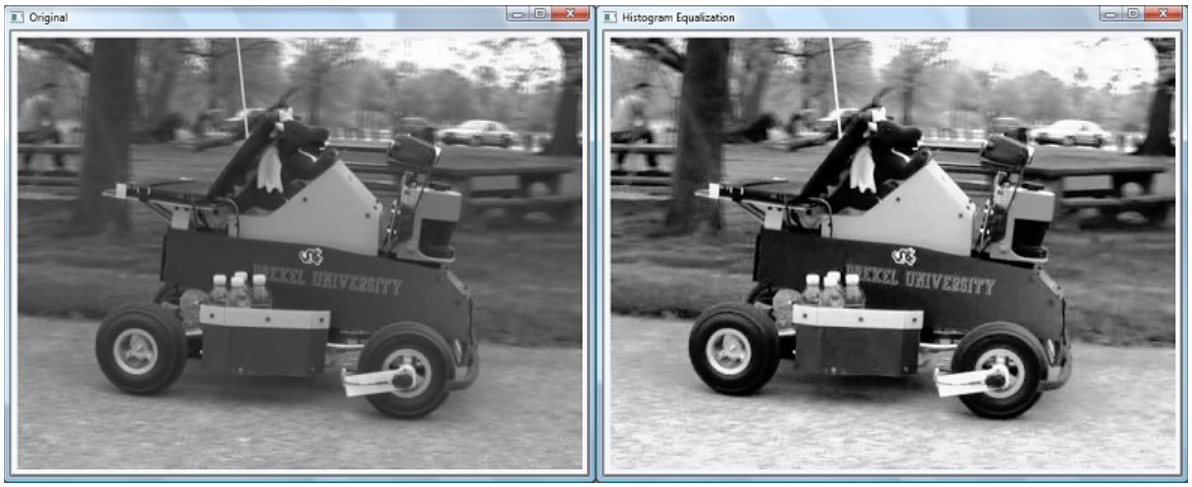
Another basic transform is an affine transform. And affine transform allows the user to warp, stretch, rotate and resize an image. Essentially the image is multiplied by 2x3 matrix to perform the transformation. An affine transform produces parallelograms (which includes standard rectangles). A more complex transform is an perspective transformation, or "Homography," which uses a 3x3 matrix and turns the image into a trapezoid (which means all affine transforms are also possible). Here I will give an example of an affine transform, read the chapter for more mathematical details and information on how to perform a more advanced perspective transform. The *cvWarpAffine* function is used to perform the transformation, this takes a similiar form to the filter functions, but the more important input here is the third input variable, the map matrix. This is what defines the transformation. For this example we use *cvGetAffineTransform* to build the parallelograming warp matrix and *cv2DRotationMatrix* to build the matrix for performing a rotation and scaling on the image. For the first, three points in the original image and corresponding points in the destination image are used to define the warp. For the second, the center, scale, and angle are used to define the rotation. This example uses sliders to allow you to change the rotation and scale. Here is the code:

```
int angle_switch_value = 0;
int angleInt = 0;
int scale_switch_value = 0;
int scaleInt = 0;

void switch_callback_a( int position ){
        angleInt = position;
}
```

```
void switch_callback_s( int position ){
	scaleInt = position;
}

int _tmain(int argc, _TCHAR* argv[])
{
	// Set up variables
	CvPoint2D32f srcTri[3], dstTri[3];
	CvMat* rot_mat = cvCreateMat(2,3,CV_32FC1);
	CvMat* warp_mat = cvCreateMat(2,3,CV_32FC1);
	IplImage *src, *dst;
	const char* name = "Affine_Transform";

	// Load image
	src=cvLoadImage("MGC.jpg");
	dst = cvCloneImage( src );
	dst->origin = src->origin;
	cvZero( dst );
	cvNamedWindow( name, 1 );

	// Create angle and scale
	double angle = 0.0;
	double scale = 1.0;

	// Create trackbars
	cvCreateTrackbar( "Angle", name, &angle_switch_value, 4, switch_callback_a );
	cvCreateTrackbar( "Scale", name, &scale_switch_value, 4, switch_callback_s );

	// Compute warp matrix
	srcTri[0].x = 0;
	srcTri[0].y = 0;
	srcTri[1].x = src->width - 1;
	srcTri[1].y = 0;
	srcTri[2].x = 0;
	srcTri[2].y = src->height - 1;

	dstTri[0].x = src->width*0.0;
	dstTri[0].y = src->height*0.25;
	dstTri[1].x = src->width*0.90;
	dstTri[1].y = src->height*0.15;
	dstTri[2].x = src->width*0.10;
	dstTri[2].y = src->height*0.75;

	cvGetAffineTransform( srcTri, dstTri, warp_mat );
	cvWarpAffine( src, dst, warp_mat );
	cvCopy ( dst, src );

	while( 1 ) {
		switch( angleInt ){
			case 0:
				angle = 0.0;
				break;
			case 1:
				angle = 20.0;
				break;
			case 2:
				angle = 40.0;
				break;
			case 3:
				angle = 60.0;
				break;
			case 4:
				angle = 90.0;
				break;
		}
		switch( scaleInt ){
			case 0:
				scale = 1.0;
				break;
			case 1:
				scale = 0.8;
				break;
			case 2:
				scale = 0.6;
				break;
			case 3:
				scale = 0.4;
				break;
			case 4:
				scale = 0.2;
```

```
                        break;
                }

                // Compute rotation matrix
                CvPoint2D32f center = cvPoint2D32f( src->width/2, src->height/2 );
                cv2DRotationMatrix( center, angle, scale, rot_mat );

                // Do the transformation
                cvWarpAffine( src, dst, rot_mat );

                cvShowImage( name, dst );

                if( cvWaitKey( 15 ) == 27 )
                        break;
        }

        cvReleaseImage( &dst );
        cvReleaseMat( &rot_mat );
        cvReleaseMat( &warp_mat );

        return 0;
}
```

Step 3: Histogram Equalization



Original and Equalized Image

This example is very simple code, but can be a very useful function. Images are captured with a limited dyanmic range due to the limitations of camera sensors. But by using histogram equalization, the brightness distribution of an image can be equalized and thereby increase the contrast of the image and in a sense increase the dynamic range. There are no extra variables for the function, *cvEqualizeHist* takes in just the source and destination image (which must be grayscale, or each color processed seperately).

```
int _tmain(int argc, _TCHAR* argv[])
{
        // Set up images
        const char* name = "Histogram Equalization";
        IplImage *img = cvLoadImage("MGC.jpg", 0);
        IplImage* out = cvCreateImage( cvGetSize(img), IPL_DEPTH_8U, 1 );

        // Show original
        cvNamedWindow( "Original", 1) ;
        cvShowImage( "Original", img );

        // Perform histogram equalization
        cvEqualizeHist( img, out );

        // Show histogram equalized
        cvNamedWindow( name, 1) ;
        cvShowImage( name, out );
```

```
        cvWaitKey();

        cvReleaseImage( &img );
        cvReleaseImage( &out );


        return 0;
}
```

Final Words

This tutorial's objective was to show how to use some image transform functions. You should be able to extend the use of these functions, and now learn about similiar more advanced function. Please read the chapter for details on functions such as the perspective transform, discrete fourier transform, and others.

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