OpenCV Tutorial 10 - Chapter 11

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

Keywords: OpenCV, computer vision, calibration, camera parameters

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

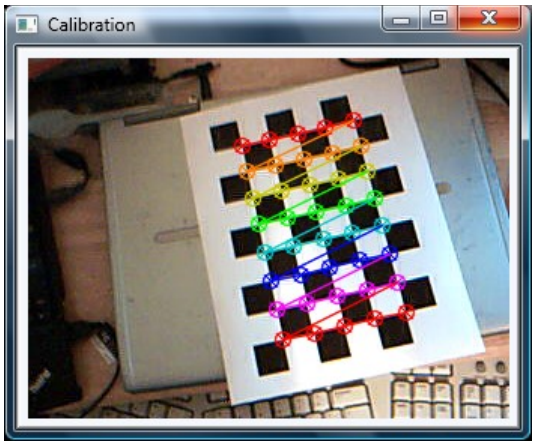
The rest of the tutorial is presented as follows:

- Step 1: Camera Calibration
- Final Words

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: Camera Calibration



Corner Finding for Calibration

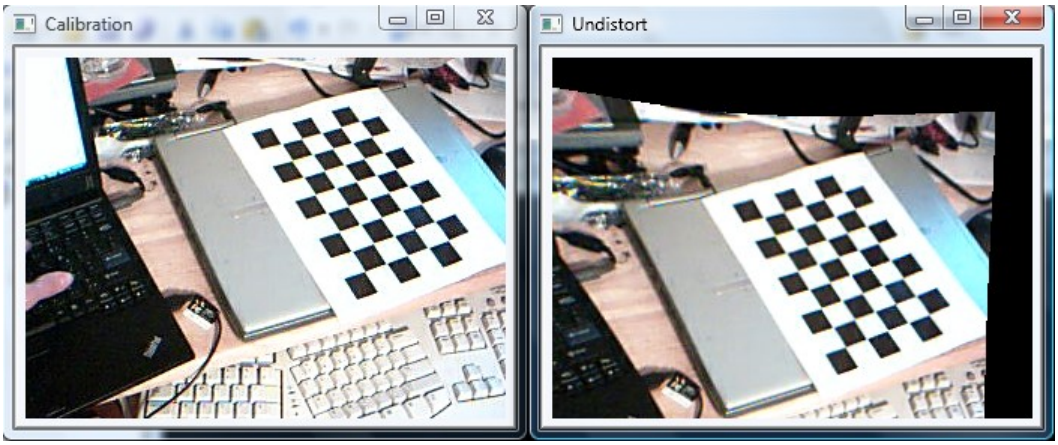
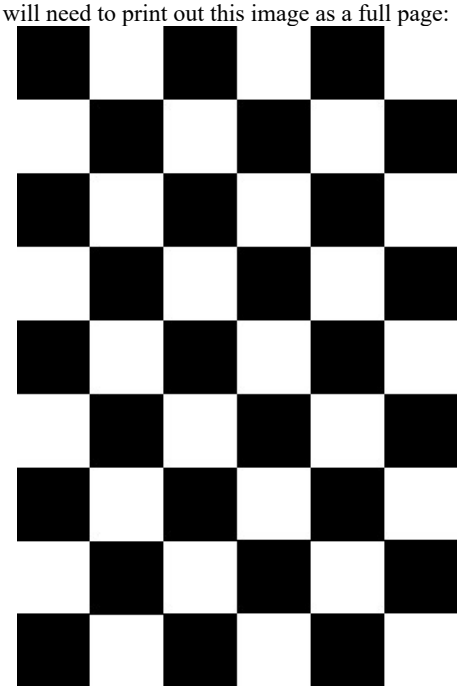


Image Undistortion After Calibration

Camera calibration is important for any image processing to be a highly accurate representation of the real world. The basics of perspective geometry are covered in the book. The concern of this tutorial is simply how to extract the relevant camera parameters from a sequence of images. For this example you will need to print out this image as a full page:



Example Checkerboard

This code uses *cvFindChessboardCorners* to find the corners, and then draw them on the current image with *cvDrawChessboardCorners*. If all the corners are successfully identified, the corners are added to image_points and object_points for later use in calibration. After all the different orientations are successfully stored (orientations are arbitrary but should have a variety of views to solve for the camera parameters), then *cvCalibrateCamera2* is used to get the camera parameters. Lastly, *cvInitUndistoryMap* is used with *cvRemap* to unwarp the camera images. So for the user, simply print out the checkerboard, then point the camera at it in various orientations as the program marks the points in each one (it is successful if a variety of colors are used, all red or nothing is a failure). The intrinsics of the camera and the distortion are stored in xml files to use in other programs. Here is the code:

```
int n_boards = 0;
const int board_dt = 20;
int board_w;
int board_h;

int _tmain(int argc, _TCHAR* argv[])
{
        board_w = 5; // Board width in squares
        board_h = 8; // Board height
        n_boards = 8; // Number of boards
        int board_n = board_w * board_h;
        CvSize board_sz = cvSize( board_w, board_h );
        CvCapture* capture = cvCreateCameraCapture( 0 );
        assert( capture );

        cvNamedWindow( "Calibration" );
        // Allocate Sotrage
        CvMat* image_points             = cvCreateMat( n_boards*board_n, 2, CV_32FC1 );
        CvMat* object_points            = cvCreateMat( n_boards*board_n, 3, CV_32FC1 );
        CvMat* point_counts                     = cvCreateMat( n_boards, 1, CV_32SC1 );
        CvMat* intrinsic_matrix         = cvCreateMat( 3, 3, CV_32FC1 );
        CvMat* distortion_coeffs        = cvCreateMat( 5, 1, CV_32FC1 );

        CvPoint2D32f* corners = new CvPoint2D32f[ board_n ];
        int corner_count;
        int successes = 0;
        int step, frame = 0;

        IplImage *image = cvQueryFrame( capture );
        IplImage *gray_image = cvCreateImage( cvGetSize( image ), 8, 1 );

        // Capture Corner views loop until we've got n_boards
```

```
// succesful captures (all corners on the board are found)

while( successes < n_boards ){
        // Skp every board_dt frames to allow user to move chessboard
        if( frame++ % board_dt == 0 ){
                // Find chessboard corners:
                int found = cvFindChessboardCorners( image, board_sz, corners,
                        &corner_count, CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS );

                // Get subpixel accuracy on those corners
                cvCvtColor( image, gray_image, CV_BGR2GRAY );
                cvFindCornerSubPix( gray_image, corners, corner_count, cvSize( 11, 11 ),
                        cvSize( -1, -1 ), cvTermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));

                // Draw it
                cvDrawChessboardCorners( image, board_sz, corners, corner_count, found );
                cvShowImage( "Calibration", image );

                // If we got a good board, add it to our data
                if( corner_count == board_n ){
                        step = successes*board_n;
                        for( int i=step, j=0; j < board_n; ++i, ++j ){
                                CV_MAT_ELEM( *image_points, float, i, 0 ) = corners[j].x;
                                CV_MAT_ELEM( *image_points, float, i, 1 ) = corners[j].y;
                                CV_MAT_ELEM( *object_points, float, i, 0 ) = j/board_w;
                                CV_MAT_ELEM( *object_points, float, i, 1 ) = j%board_w;
                                CV_MAT_ELEM( *object_points, float, i, 2 ) = 0.0f;
                        }
                        CV_MAT_ELEM( *point_counts, int, successes, 0 ) = board_n;
                        successes++;
                }
        }

        // Handle pause/unpause and ESC
        int c = cvWaitKey( 15 );
        if( c == 'p' ){
                c = 0;
                while( c != 'p' && c != 27 ){
                        c = cvWaitKey( 250 );
                }
        }
        if( c == 27 )
                return 0;
        image = cvQueryFrame( capture ); // Get next image
} // End collection while loop

// Allocate matrices according to how many chessboards found
CvMat* object_points2 = cvCreateMat( successes*board_n, 3, CV_32FC1 );
CvMat* image_points2 = cvCreateMat( successes*board_n, 2, CV_32FC1 );
CvMat* point_counts2 = cvCreateMat( successes, 1, CV_32SC1 );

// Transfer the points into the correct size matrices
for( int i = 0; i < successes*board_n; ++i ){
        CV_MAT_ELEM( *image_points2, float, i, 0) = CV_MAT_ELEM( *image_points, float, i, 0 );
        CV_MAT_ELEM( *image_points2, float, i, 1) = CV_MAT_ELEM( *image_points, float, i, 1 );
        CV_MAT_ELEM( *object_points2, float, i, 0) = CV_MAT_ELEM( *object_points, float, i, 0 );
        CV_MAT_ELEM( *object_points2, float, i, 1) = CV_MAT_ELEM( *object_points, float, i, 1 );
        CV_MAT_ELEM( *object_points2, float, i, 2) = CV_MAT_ELEM( *object_points, float, i, 2 );
}

for( int i=0; i < successes; ++i ){
        CV_MAT_ELEM( *point_counts2, int, i, 0 ) = CV_MAT_ELEM( *point_counts, int, i, 0 );
}
cvReleaseMat( &object_points );
cvReleaseMat( &image_points );
cvReleaseMat( &point_counts );

// At this point we have all the chessboard corners we need
// Initiliazie the intrinsic matrix such that the two focal lengths
// have a ratio of 1.0

CV_MAT_ELEM( *intrinsic_matrix, float, 0, 0 ) = 1.0;
CV_MAT_ELEM( *intrinsic_matrix, float, 1, 1 ) = 1.0;

// Calibrate the camera
cvCalibrateCamera2( object_points2, image_points2, point_counts2, cvGetSize( image ),
        intrinsic_matrix, distortion_coeffs, NULL, NULL, CV_CALIB_FIX_ASPECT_RATIO );

// Save the intrinsics and distortions
cvSave( "Intrinsics.xml", intrinsic_matrix );
```

```
        cvSave( "Distortion.xml", distortion_coeffs );

        // Example of loading these matrices back in
        CvMat *intrinsic = (CvMat*)cvLoad( "Intrinsics.xml" );
        CvMat *distortion = (CvMat*)cvLoad( "Distortion.xml" );

        // Build the undistort map that we will use for all subsequent frames
        IplImage* mapx = cvCreateImage( cvGetSize( image ), IPL_DEPTH_32F, 1 );
        IplImage* mapy = cvCreateImage( cvGetSize( image ), IPL_DEPTH_32F, 1 );
        cvInitUndistortMap( intrinsic, distortion, mapx, mapy );

        // Run the camera to the screen, now showing the raw and undistorted image
        cvNamedWindow( "Undistort" );

        while( image ){
                IplImage *t = cvCloneImage( image );
                cvShowImage( "Calibration", image ); // Show raw image
                cvRemap( t, image, mapx, mapy ); // undistort image
                cvReleaseImage( &t );
                cvShowImage( "Undistort", image ); // Show corrected image

                // Handle pause/unpause and esc
                int c = cvWaitKey( 15 );
                if( c == 'p' ){
                        c = 0;
                        while( c != 'p' && c != 27 ){
                                c = cvWaitKey( 250 );
                        }
                }
                if( c == 27 )
                        break;
                image = cvQueryFrame( capture );
        }

        return 0;
}
```

Final Words

This tutorial's objective was to show how to extract camera parameters using calibration routines.

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