A tool with improved accuracy in checker-board corner detection and automatic data processing for vision-based measurement instrument (camera) Calibration

Traditional camera calibration methods compute camera intrinsic parameters from a set of target features with known geometries. The most common target is a checkerboard. Calibration usually requires users to select corners and areas on the checkerboard images manually. The manual process has several problems:

- Time-consuming. Users must manually click four corners on each checkerboard image to define the region of interest. When there are multiple images, this process is very time-consuming.
- Prone to error. The corner detection relies on the accuracy of the user's clicking which is prone to error.
- The difficulty for stereo camera calibration. Using the dual camera as an example, stereo camera calibration needs to find correspondence between the left and right camera images. Users either have to make sure both cameras contain the full checkerboard image, or count and align the same corners for both left and right images manually. This process is tedious and time-consuming.

A method and tool for automatic camera calibration were developed. This method has advantages: 1) it doesn't involve user clickings to define the region of interest, which eliminates the error and accuracy problems from human operation; 2) Images are processed automatically in batch, eliminating the time-consuming manual data processes; 3) improved the accuracy by developing advanced algorithms in the checker-board corner detection, thus improved the vision-based system's accuracy; 4) Enabled the creation of user-defined markers for auto data processing. The key to the automatic method is to enable auto-counting and auto-alignment. As shown in Fig. 1, the calibration board produced by Cognex has two long rectangle makers in the center. The special markers are utilized as the indicator of the checkboard's orientation and the center position. If the marker features are correctly detected, any corner in the image can find the correspondence to the center. Users may customize their markers on their checkerboard, for example, by adding extra dots or special shapes as indicators. This tool can help to improve the accuracy and efficiency in vision-based system calibration, where cameras and vision-based systems are widely used in machine vision, robot guidance, inspection, and health condition monitoring for industrial applications.

Figure 1. An example of calibration board with center markers

Tool description:

This program is developed using Python 3.

Setup for Windows

- Install anaconda Python environment
- Install the following packages through conda or pip: numpy, matplotlib, python- opencv, scipy

Single Camera Calibration

Script to run: CalibrateSingleCamera. y

Please note that all parameters are hardcoded.

- You will need to specify the BOARDHEIGHT and BOARDWIDTH, these are the number of squares in rows/columns for the calibration board. Most of the time you don't need to change these numbers if the calibration board is not changed.
- zoom_factor: This is used to estimate the projection back to frontal parallel for iterated calibration, should be left at 0.8-1 most of the cases.
- iteration: number of times to iteratively calibrate the camera. The current take is that we don't see a lot of benefit by iterating more than 1 time.
- resize_factor: In theory, if we upsample the image, we may get better results for corner detection. Currently, it is left at 1.
- calib_img_path: the folder contains all the calibration images. It should have one ofthe following formats: png, bmp, tif, or jpg.

The results are saved in yaml files.

Stereo Camera Calibration

Script to run: CalibrateStereoCamera. y

In addition to the parameters shown in single camera calibration, You will need to provide the following:

- **calib_left_img_path** and **calib_left_img_path.**

The results are saved in yaml files.

Algorithm Procedures

1. Obtain negative images.

Negative images (as shown in Figure 2b) are used because our approach is based on detection and fitting contour to ‘black’ squares and rectangles.

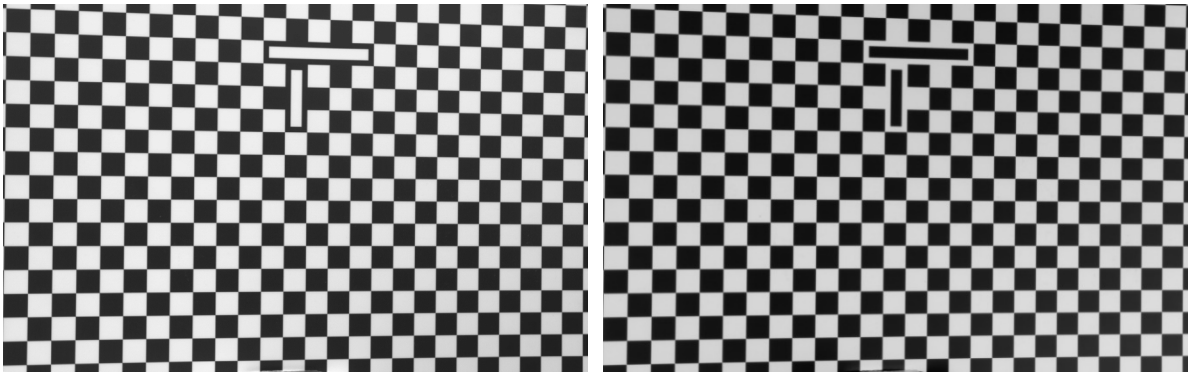

Figure 2. Original calibration image (a) on the left and negative image (b) on the right.

2. Obtain binarized image.

3. Squares and rectangles detection.

4. Recursive algorithm to search and map the squares to a grid.

Fig. 3 shows the valid contours detected. Fig. 4 shows the labeled corners.

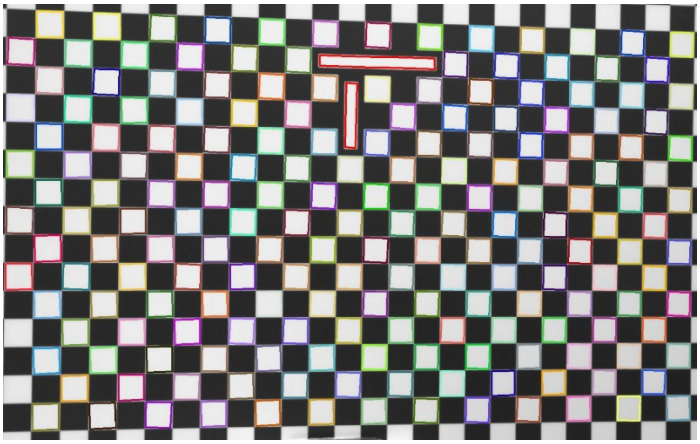

Figure 3. Illustration of the valid contours, squares are shown in random color, long rectangles are shown in red color.

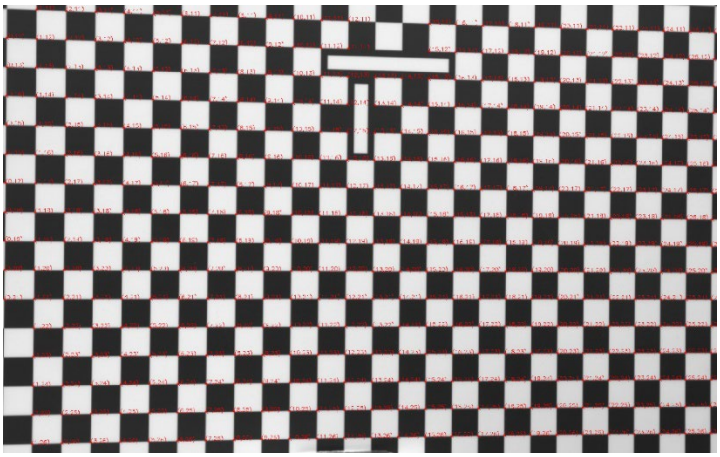

Figure 4. Labeled corners

5. Subpixel edge detection

The detected corner was passed through a subpixel edge detection module to iterate to find the sub-pixel accurate location of corners or radial saddle points, as shown in Fig. 5.

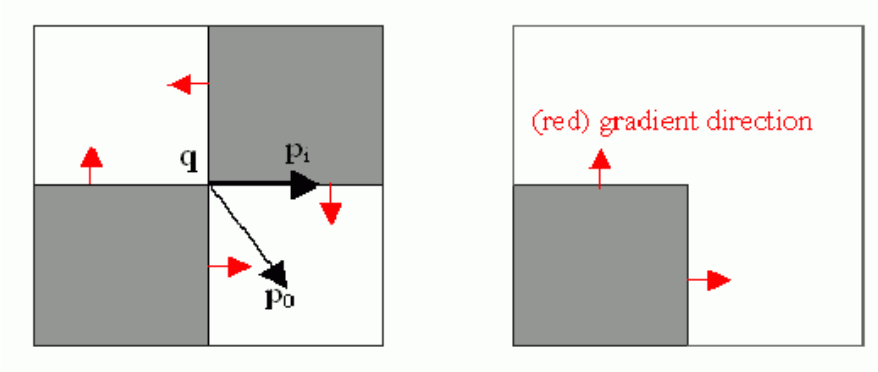

Figure 5. Subpixel edge detection algorithm from OpenCV library

6. Align the grids and corner detection refinement.

Final calibration results

The table below shows the reprojection errors before and after the subpixel improvement. Fig. 6 shows the reprojection error after calibration.

Calibration results (using saddle point subpixel detection)	Calibration results (combing Harris corner improvement)
1 sigma reprojection error: 0.144 Pixel	1 sigma reprojection error: 0.130 Pixel

Final calibration of the single camera

Intrinsic Matrix:[[4.62135558e+03 0.00000000e+00 9.39734834e+02]

[0.00000000e+00 4.62093465e+03 5.85333167e+02]

[0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Distortion:

[1.13627163e-01, 5.70960007e-01, 2.45515559e-03,-1.38852863e-03, -1.17335534e+01]]

Figure 6. Reprojection error for each of the 40 calibration images.