FIATAL MŰSZAKIAK TUDOMÁNYOS ÜLÉSSZAKA XIX.

Kolozsvár, 2014. március 20–21.

KAMERA KALIBRÁCIÓ OPENCV HASZNÁLATÁVAL NOTEBOOK CAMERA CALIBRATION USING OPENCV

Guilherme M. PEREIRA⁽¹⁾, Armando ALBERTAZZI G.Jr.⁽²⁾,
Maryah E.M. HAERTEL⁽²⁾

(1) CAPES Foundation, Ministry of Education of Brazil, Address: 70.040-020, Brazil, Brasilia – DF, guilherme.mp.eng@gmail.com.

(2) LABMETRO (Federal University of Santa Catarina, Technologic Center, Mechanical Engineering Department) Address: 88040-970, Brazil, Florianópolis-SC, Caixa Postal: 5053; Telefon: +55 48 3239-2030 / Fax: +55 48 3239-2009, albertazzi@labmetro.ufsc.br, mmh@labmetro.ufsc.br.

Abstract

This paper is concerned about the implementation of a software in C# that calculates the calibration of a notebook camera based on Zhang's method and using the tools provided by the OpenCV (Open Source Computer Vision Library). First it is given a brief introduction to calibration of cameras, focusing in the most important calculations and in the concept involved on it, then the EmguCV wrapper is introduced to justify its use since the OpenCV library needs it to be accessed by software implemented in C# and finally the paper provides detailed information about the implementation, emphasizing in how to use the calibration camera methods of this powerful computer vision tool. After the implementation of the camera calibration system some tests were made and a section in the end of this paper was created to discuss the results. Different configurations were tested and through them it was possible to take some conclusions and propose improvements for the software.

Keywords: camera calibration, OpenCV, EmguCV.

Összefoglaló

Ez az írás aggodnik végrehajtásáról szoftver C # , amely kiszámítja a kalibrációs egy notebook kamera alapján Zhang módszer és eszközeivel által biztosított OpenCV (Open Source Computer Vision Library) . Először is ez adott egy rövid bevezetés a kalibrálás kamerák , összpontosítva a legfontosabb számításokat , és a fogalom rajta , akkor a EmguCV wrapper be jogosítanak fel , mivel a OpenCV könyvtárban kell , hogy legyen elérhető szoftver végrehajtott C # és végül a papírt részletes információkkal szolgál a végrehajtás , hangsúlyozva , hogyan kell használni a kalibrációs kamera módszereit ezt a hatalmas számítógépes látás eszköze. Miután a végrehajtás a kamera kalibrációs rendszer néhány vizsgálatot végeztek , és egy szakaszt a végén ez a dokumentum jött létre , hogy bemutassa a kvantitatív eredményeket , valamint megvitatja és értékeli azokat . A különböző konfigurációk tesztelték , és rajtuk keresztül lehetett , hogy bizonyos következtetéseket és javaslatokat tenni a szoftver.

Kulcsszavak: kamera kalibráció, OpenCV, EmguCV.

1. Introduction

Camera calibration is a very important step in 3D computer vision in order to extract metric information from 2D images [1]. There are some applications to calibrate cameras and one of them is a Matlab toolbox [2] that can provide quite good and reliable results but its process sometimes requires a lot of efforts for the user, moreover the steps of capturing

images, calibration and exportation of the results are not totally integrated. Therefore the software discussed in this paper has the aim of provide an easy and automated solution to calibrate a notebook camera as well as export the results. The software was developed in C# by using Microsoft Visual Studio 2010 IDE (Integrated Development Environment) under .NET Framework 4 and the EmguCV [3], which is a wrapper to the OpenCV library [4] because this one does not have a direct interface to C# language. Camera calibration consists in finding internal parameters of the camera that affect image processing like: lens distortions, focal length, coordinates of the central point of the image, scaling factors for row pixels and column pixels [5]. The calibration discussed in this paper was based on the mathematic model of a *pinhole camera* [6] and its equation can be written like in (1):

$$\mathbf{x} = \mathbf{KR}[\mathbf{I} | -\tilde{\mathbf{C}}]\mathbf{X} \quad \left\{ \begin{matrix} \lambda u \\ \lambda v \\ \lambda \end{matrix} \right\} = \mathbf{K}[\mathbf{I} | \mathbf{0}] \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\mathbf{C}} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{Bmatrix} X \\ Y \\ Z \\ 1 \end{Bmatrix} = \mathbf{K} \mathbf{R}[\mathbf{I} | -\tilde{\mathbf{C}}] \begin{Bmatrix} X \\ Y \\ Z \\ 1 \end{Bmatrix} \quad (1)$$

Where: (u,v) are the image coordinates, (X,Y,Z) are the 3D coordinates, $\lambda = Z$, $\mathbf{K}$ is the *intrinsic parameters matrix*; $\mathbf{R}_{3 \times 3}$ and $\tilde{\mathbf{C}}$ correspond to the *extrinsic parameters* containing the rotational and translational values of the camera in relation to the global reference frame. So by using the projection matrix or calibration matrix denoted by $\mathbf{P} = \mathbf{K} \mathbf{R}[\mathbf{I} | -\tilde{\mathbf{C}}]$, it is possible to know the correspondent 3D points in space, given the 2D points in the image and vice versa.

2. Implementation

The implementation of the calibration system was divided in steps and they are described in the following subsections.

2.1 Capturing images

First of all it was implemented the functionalities of access the notebook camera and make the automatic acquisition of the images containing the pattern (a chessboard) in different positions. The EmguCV (installed version: 2.4.2) provides an example in the installation directory called *CameraCapture*, where the project shows a simple implementation of capturing frames of the notebook camera. So the class *Capture* was used to get the frames [7] and the Windows Form called *ImageBox Control* was used to show the current frames [8] and all of them are provided by the EmguCV. As aborded in [1] it is possible to calibrate the camera with the minimum of 2 images but a number between 7 and 12 images can produce better results. The images were captured in 640x480 pixels, converted to gray scale and saved in JPG format.

2.2 Finding Chessboard Corners

After capturing a set of images they must be processed and the corners coordinates extracted. The method *FindChessBoardCorners* provided by EmguCV can be used to extract the corners coordinates (u,v) in pixels, of an image passed as parameter and these information is saved in a variable also passed as parameter. The documentation of this method can be found in [7] and in [9] it is possible to find a simple tutorial describing the use of this method

2.3 Calculating intrinsic parameters, extrinsic parameters and calibration matrix

The next step after having the coordinates of the chessboard corners is to calculate the intrinsic parameters, the extrinsic parameters and the calibration matrix. In [10] it is possible

to find a nice method called *CalibrateCamera* that receive parameters and use them in another method called *cvCalibrateCamera2* [7] that estimate the intrinsic and extrinsic parameters (also provided by the EmguCV). After this processing the *calibration matrix*[3x4] can be calculated by multiplying the matrix of intrinsic parameters [3x3] and the matrix of extrinsic parameters [3x4].

2.4 Projecting Corners

The calibration matrix can be used to project 2D from 3D points so in this next step the main aim is to use it to transform the 3D points of the chessboard corners to 2D coordinates and calculate the errors with the points found by the method *FindChessBoardCorners*. The EmguCV has a method called *cvProjectPoints2* to do it and its description can be found in [7] but the method *ProjectPoints2* (its code can be found in [10]) was implemented to make it easier to use first one.

2.5 Calculating Calibration Error

By now we have the 2D coordinates calculated by the methods *FindChessBoardCorners* and *ProjectPoints2* and with these informations it is possible to have the errors between the corners. So the method *CalculateErrorBetweenCorners* was implemented with this aim and the results were exported to an Excel file to make it easier to analyze them. After all the calculations the results are exported to an Excel file so the user can analyze the obtained values and make conclusions about the calibration.

2.6 Exporting the Results

To export the results to an Excel file it is not necessary to download any package but only add the following reference: *Add Reference-> .NET-> Microsoft.Office.Interop.Excel*. After that, insert the following code and use Excel to refer to the application and edit the file: *using Excel = Microsoft.Office.Interop.Excel*. The examples shown in [13] and [14] helped in the construction of the results exportation method.

2.7 Reducing the Calibration Errors

An improvement was applied to the algorithm with the aim of reduce the calibration errors. The basic idea was insert iterations to the process and repeat the following steps: take the corners calculated for the method *ProjectPoints2*, use them in the method *CalibrateCamera*, project the points and calculate the errors again. The number of iterations can be chosen by the user but some tests were made and 2 iterations are good enough to reduce the calibration errors.

3. Tests and Results

The tests were made using the minimum of 7 images and the maximum of 12 images and the aim of the *first test* was to evaluate the *global error* and the *standard deviation* in relation to different number of captured images used in the calibration. Multiple calibrations were made varying the number of captured images. For this first test the global errors values do not show results that allow us make conclusions but in relation to the standard deviations, calibrations using 7, 8 and 12 images have generated small values therefore better results. In most cases the global errors and the standard deviations without considering the distortion coefficients have smaller values and it can be explained by the fact that the lens has small distortions and most of the corners are positioned in the center of the images where this problem doesn't affect too much the results. The *second test* was made to evaluate the repetitivity of the software. The same images were processed 5 times and the same results were generated, what proves that the algorithm of the OpenCV is reliable.

4. Conclusion

According to the proposed software defined in the beginning of the implementation all the aims were achieved and the calibration system has proved to be practical and easy to be operated by the user. Although the methods provided by the EmguCV have been implemented successfully it was difficult to learn how to use them because the parameters required had to be passed in very specific manners beyond that the documentation for C# was a little bit scarce. Fortunately there were some codes in the internet which helped a lot in the understanding and use of the methods but lots of tests were necessary to compose the final solution. There are some additional improvements that could be implemented in this solution:

- Create an exclusion criterion to discard the images with big errors in the identification of the corners so they won't affect the calibration and errors calculations;
- Adapt the software to other cameras and evaluate the results. Lens with more distortions can be used to analyze the results.

5. Acknowledgements

The authors thank CAPES (*Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*) and the Science Without Borders program for the scholarship given to Guilherme Martins Pereira under process number 88888.016879/2013-00.

References

- [1] Zhang, Z.: *A flexible new technique for camera calibration*. IEEE Trans. Pattern Anal. Machine Intell. vol.22 n.11, p. 1330–1334, 2000.
- [2] *Calibration using a MATLAB toolbox*: http://www.vision.caltech.edu/bouguetj/calib_doc/. Last access: 03/03/2014.
- [3] *Main page of EmguCV*: http://www.emgu.com/wiki/index.php/Main_Page. Last access: 03/03/2014.
- [4] *About OpenCV*: <http://opencv.org/about.html>. Last access: 03/03/2014.
- [5] *Camera calibration*: <http://www.umi.acs.umd.edu/~ramani/cmssc828d/lecture9.pdf>. Last access: 03/03/2014.
- [6] *Pinhole camera*: http://en.wikipedia.org/wiki/Pinhole_camera. Last access: 03/03/2014.
- [7] *EmguCV documentation*: <http://www.emgu.com/wiki/files/2.4.2/document/Index.html>. Last access: 03/03/2014.
- [8] *ImageBox Control*: http://www.emgu.com/wiki/index.php/Add_ImageBox_Control. Last access: 03/03/2014.
- [9] *FindChessBoardCorners tutorial*: <http://mymobilerobots.com/myblog/academic/emgu-cv-c-tutorial-find-chess-board-corners/>. Last access: 03/03/2014.
- [10] *Calibration methods implemented*: http://read.pudn.com/downloads185/sourcecode/windows/869647/src/Emgu.CV/CameraCalibration/CameraCalibration.cs_.htm. Last access: 03/03/2014.
- [11] *Initializing the objectPoints variable*: <http://www.aishack.in/2010/07/calibrating-undistorting-with-opencv-in-c-oh-yeah/>. Last access: 03/03/2014.
- [12] *Standard deviation*: http://en.wikipedia.org/wiki/Standard_deviation. Last access: 03/03/2014.
- [13] *How to write in an Excel file (code 1)*: <http://social.msdn.microsoft.com/Forums/en-US/csharpgeneral/thread/ef11a193-54f3-407b-9374-9f5770fd9fd7>. Last access: 03/03/2014.
- [14] *How to write in an Excel file (code 2)*: <http://stackoverflow.com/questions/9769703/exporting-to-xlsx-using-microsoft-office-interop-excel-saveas-error>. Last access: 03/03/2014.