Calibración de una cámara haciendo uso de funciones de OPENCV

Trabajo práctico correspondiente a la cátedra: Visión por Computadora. Ingeniería electrónica. Año 2020

Coronel Martín, legajo: 69419, Fantin Stéfano, legajo: 70502
Universidad Tecnológica Nacional, FRC
Córdoba, Argentina
martin97coronel@gmail.com
fantinstefano96@gmail.com

***Abstract*—En este informe explayaremos el método de corrección de distorsión de las imágenes tomadas por una cámara fotográfica, se extraerá la matriz de transformación y corregirán las imágenes tomadas.**

***Index Terms*—distorsion, opencv, calibration, Computer Vision, UTN**

I. Introducción

Es común que las lentes utilizadas en cámaras fotográficas introduzcan distorsión en las imágenes tomadas. En nuestro caso nos centraremos en la distorsión radial, la cual transforma las líneas rectas del plano real en líneas curvas en el plano de la imagen y se acentúa a medida que nos distanciamos del centro de la toma y en la distorsión tangencial, la cual es causada por la lente cuando no es paralela al plano de la imagen. Se puede observar en la Figura 1 cómo la captura de la imagen transforma el borde del tablero de ajedrez, el cual es recto, en una curva que no coincide con la línea roja auxiliar dibujada.

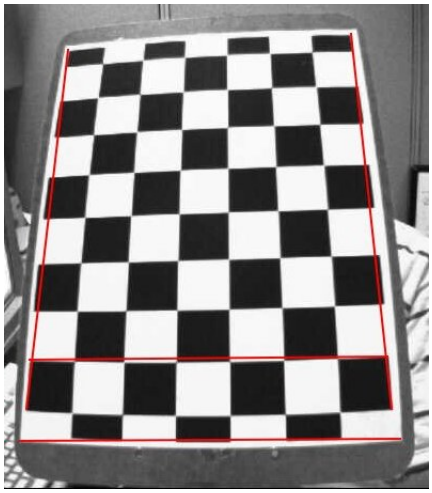

Figure 1. Líneas rectas en tablero de ajedrez que por distorsión se perciben como curvas

II. Desarrollo

Según la naturaleza de las distorciones a quitar (radial y tangencial), debemos encontrar dos matrices que nos permitan realizar este trabajo. Los coeficientes de distorsión:

$$\begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{1}$$

Y para la conversión de unidades utilizaremos la siguiente matriz:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2}$$

La presencia de w y Z en la matriz anterior se debe a que el sistema fue llevado a coordenadas homogéneas.
f_x y f_y son las distancias focales físicas de la cámara, mientras que c_x y c_y son los centros ópticos expresados en pixeles.

III. Preparación

Llamamos calibración al proceso de averiguar las matrices (1) y (2). Para este proceso haremos uso de funciones que nos brinda la biblioteca de visión artificial opencv.
Necesitamos disponer de un conjunto de tomas de un patrón de calibración, en nuestro caso un tablero de ajedrez de $9x6$, colocado en diferentes ubicaciones y orientaciones.
Para poder hacer el cálculo de las matrices debemos poder correlacionar el poscionamiento en el espacio de los puntos de intersección del cuadriculado del tablero con los mismos puntos en el plano bidimensional de la imagen.
Los puntos de intersección en el tablero de ajedrez se pueden encontrar con la función $cv.findChessboardCorners()$, a la que debemos pasarle como argumento la imagen donde deseamos encontrar los puntos de intersección y una tupla con la cantidad de esquinas internas del tablero utilizado. Nos devuelve los puntos de las esquinas y ret (retorno) positivo si encontró el patrón.
Con el método $cv.cornerSubPix()$ podemos aumentar la precisón de las esquinas. Esta función recibe la imagen en escala de grises, las esquinas anteriormente detectadas con $cv.findChessboardCorners()$, la ventana de búsqueda y una tupla "criteria". Esta tupla representa los criterios de iteración del algoritmo, como la cantidad de iteraciones maximas y el error máximo admitido.
Obtenidas las esquinas refinadas las guardamos en un objeto y dibujamos el patrón usando la función

$cv.drawChessboardCorners()$,como se puede ver en la figura 2.

Figure 2. Patron encontrado por la función $cv.findChessboardCorners()$

IV. Calibración

Una vez obtenidos los puntos de interseción del tablero podemos realizar la calibración. Para esto usamos la función $cv.calibrateCamera()$ que recibe como parámetro los puntos del objeto 3D y sus correspondientes proyecciones 2D en cada vista, además del tamaño de la imagen. La función genera y devuelve la matriz intrínseca de la cámara, el vector de coeficientes de distorsión y los vectores de rotación y traslación.

Una vez obtenidos estos parámetros, estamos listos para transformar las imágenes compensando la distorsión de la lente. Para esto usamos la función $cv.undistort()$ que recibe como parámetros la imagen distorsionada (Figura 3 A), la matriz de la cámara y el vector de coeficientes de distorción. Este método nos devuelve la imagen sin distorción, como se puede ver en la figura 3 B.

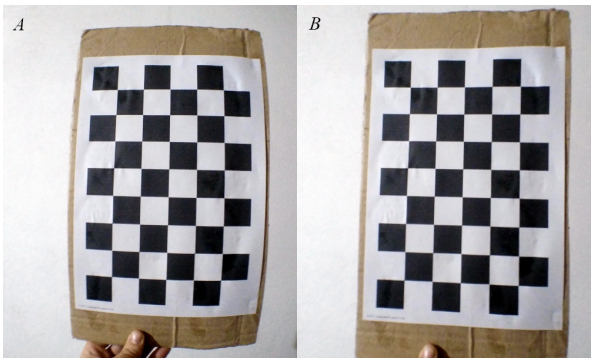

Figure 3. A: imagen original. B: imagen corregida

V. Conclusión

Gracias a las librerias de OpenCV logramos conseguir de forma sencilla los parámetros de calibración de la cámara para luego corregir la distorción de las imágenes. Podemos concluir que los resultados obtenidos fueron excelentes, ya que se puede notar a simple vista cómo se corrige el patrón cuadriculado del tablero.

References

[1] Redolfi Javier A. And Araguas Roberto Gaston, *Filminas de la Catedra "Vision por Computadora"*.
[2] Peter Todorov, *Multi-camera Calibration*.
[3] Open CV, *https://docs.opencv.org/master/dc/dbb/tutorial$_p y_c alibration.html$*.

VI. Apendice

```
#coding=utf-8

import glob
import numpy as np
import cv2
import os


criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

objp = np.zeros((np.prod((8, 5)), 3), np.float32)
objp[:, :2] = np.indices((8, 5)).T.reshape(-1, 2)


objpoints = []
imgpoints = []

images = glob.glob('chess/*.png')

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    ret, corners = cv2.findChessboardCorners(gray, (8, 5), None)

    if ret is True:

        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray, corners, (11, 11),  (-1, -1), criteria)
        imgpoints.append(corners2)
        name = os.path.relpath(fname, 'chess')
        img = cv2.drawChessboardCorners(img, (8, 5), corners2, ret)
        cv2.imwrite('chess/draw_' + name, img)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

print("\n", dist, "\n" , mtx)
for fname in images:

    img = cv2.imread(fname)
    name = os.path.relpath(fname, 'chess')
    cv2.imwrite('chess/un_' + name, cv2.undistort(img, mtx, dist))
```