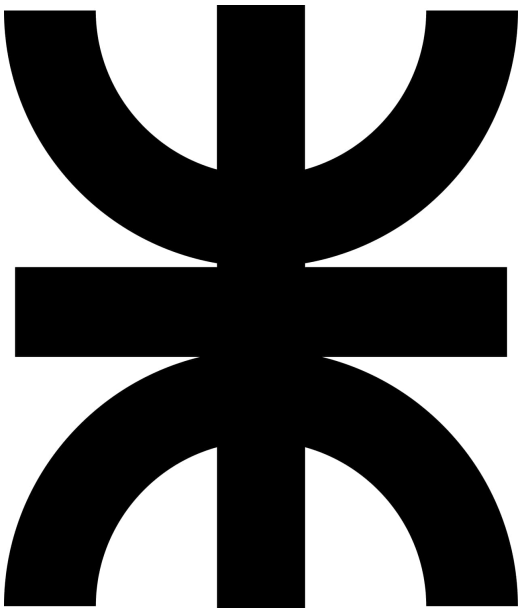
Trabajo Práctico N°10

Calibración de una cámara

Curso 6R3



Cajal, Esteban Natanael	66950
De Luca, Lautaro	66116
Fuentes, Juan José	63715

Visión Por Computadora
Universidad Tecnológica Nacional
Facultad Regional Córdoba
31 de julio de 2020

Índice

1. Introducción

En este informe se desarrollará el funcionamiento del código del Trabajo Práctico 10, el cual al darle una serie de imágenes de un patrón con dimensiones conocidas capturadas con una cámara que sabemos que presenta cierta distorsión, el mismo encontrará los coeficientes de distorsión (en este caso radial) y generará una imagen nueva sin la misma.

2. Consigna

- Utilizar los códigos de arriba para calibrar una cámara.
- Usar pickle para guardar la matriz de calibración y los coeficientes de distorsión.
- Para corroborar la correcta calibración usar el resultado de calibración obtenido para eliminar la distorsión de una imagen.

3. Código

En nuestro caso dividimos el código en dos para que tenga sentido el uso de serializar y deserializar los datos de los coeficientes de distorsión y la matriz de cámara.

```
import numpy as np

import cv2
import glob
import pickle

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

objp = np.zeros((8*11,3), np.float32)
objp [:,:2] = np.mgrid [0:11,0:8].T.reshape( -1 , 2)

objpoints = []
imgpoints = []

images = glob.glob('tmp/*.jpg')

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret,corners = cv2.findChessboardCorners(gray,(11,8), None)
    if ret is True:
        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray,corners,(11,11) , (-1 ,-1) , criteria)
        imgpoints.append(corners2)
        img = cv2.drawChessboardCorners(img,(11,8) ,corners2, ret)
        cv2.imshow('img', img)
        cv2.waitKey(300)
cv2.destroyAllWindows()

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

with open ('cameraMtx.pickle', "wb") as matrix_file:
    pickle.dump(cameraMtx, matrix_file, pickle.HIGHEST_PROTOCOL)
with open ('distCoeffs.pickle', "wb") as coeffs_file:
    pickle.dump(distCoeffs, coeffs_file, pickle.HIGHEST_PROTOCOL)
```

Listing 1: Parte 1

```python
import numpy as np

import cv2
import glob
import pickle

images = glob.glob('tmp/*.jpg')
correctedImg = []
i=0

with open ('cameraMtx.pickle', "rb") as cameraMtx_file:
    cameraMtx = pickle.load(cameraMtx_file)

with open ('distCoeffs.pickle', "rb") as distCoeffs_file:
    distCoeffs = pickle.load(distCoeffs_file)

for fname in images:
    img = cv2.imread(fname)
    correctedImg = cv2.undistort(img, cameraMtx, distCoeffs, correctedImg)
    cv2.imwrite('{:>2}.jpg'.format(i), correctedImg)
    i+=1
```

Listing 2: Parte 2

4. Desarrollo

Trabajaremos sobre un grupo de 19 imágenes de un patrón tomadas por una cámara con distorsión como la ilustrada en la figura 1.

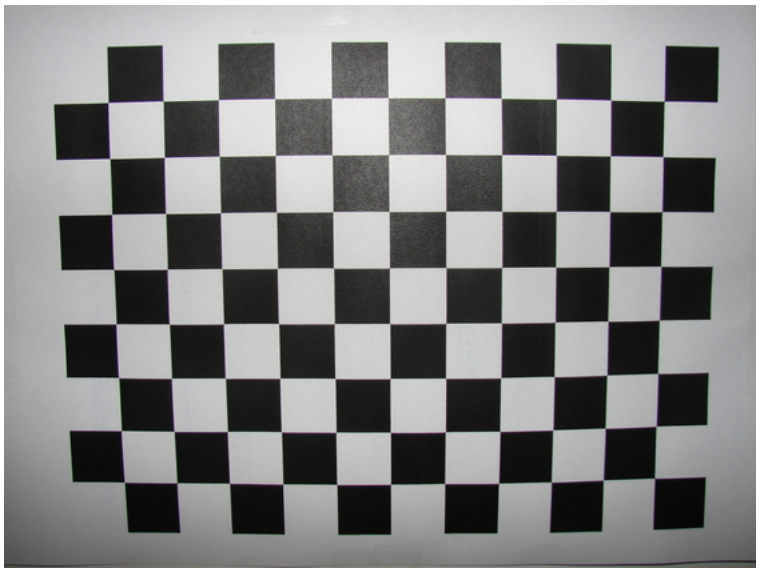


Figura 1: Imagen del patrón utilizada.

Primero se establecen los criterios de iteración del algoritmo, ya que al ser un algoritmo iterativo debemos definir la cantidad de iteraciones máximas y el error máximo admitido para así hacer un algoritmo más eficiente. En este caso se eligió una cantidad máxima de 30 iteraciones y un error máximo de 0.001 píxeles.

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

Luego inicializaremos nuestro arreglo de puntos en 3D a la cual llamaremos objp, este será de 8 x 11 (ya que esas son las esquinas que posee nuestro patrón) y tendrá 3 elementos cada uno. En este arreglo guardaremos la ubicación de cada una de las esquinas.

Después con la funcion mgrid (mesh grid) de numpy generaremos todas las posibles combinaciones de pares coordenados de nuestras esquinas y los guardaremos en los primeros dos elementos de cada uno de los espacios de nuestro arreglo objp.

```
objp = np.zeros((8*11,3), np.float32)
objp [:,:2] = np.mgrid [0:11,0:8].T.reshape( -1 , 2)
```

Con el paquete "glob"que nos permite leer de manera fácil todas las imágenes que tenemos guardadas en nuestro directorio, creamos una variable "images".

```
images = glob.glob('tmp/*.jpg')
```

Luego recorremos nuestras imágenes con un for, cada iteración del mismo "fname" tendrá el nombre de la imagen subsiguiente hasta terminar nuestro conjunto de imágenes. Leemos la imagen con un imread de la librería opencv, la convertimos a escala de grises, encontramos las esquinas con el método findChessboardCorners, y luego si las encontramos, en el arreglo objpoints agregamos los puntos que generamos anteriormente.

Con el método "cornerSubPix"mejoramos la detección de esquinas y le vamos a pasar la imagen en escala de grises, las esquinas anteriormente detectadas con findChessboardCorners, la ventana de búsqueda y el criterio, nos va a devolver las esquinas refinadas" en una variable llamada corners2. Posteriormente depositaremos las coordenadas de estas esquinas en nuestro arreglo "imgpoints", y dibujaremos las esquinas detectadas sobre la foto con el método drawChessboardCorners. Mostraremos la imagen en pantalla mas que nada para tener una referencia visual de lo que está haciendo el programa y poder revisar que lo que hicimos está bien.

```
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret,corners = cv2.findChessboardCorners(gray,(11,8), None)
    if ret is True:
        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray,corners,(11,11) , (-1 ,-1) , criteria)
        imgpoints.append(corners2)
        img = cv2.drawChessboardCorners(img,(11,8) ,corners2, ret)
        cv2.imshow('img', img)
        cv2.waitKey(300)
cv2.destroyAllWindows()
```

Por último generaremos nuestros coeficientes de distorsión y nuestra matriz de cámara con el método de opencv calibrateCamera, al cual le pasaremos los arreglos generados anteriormente "objpoints" y "imgpoints", y por último la imagen en escala de grises.

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

De los datos que nos devuelve la función calibrateCamera tomaremos la matriz de cámara y los coeficientes de distorsión y los serializaremos haciendo uso del paquete pickle, generando un "dump"de nuestros datos y guardándolo como archivo de extensión ".pickle".

```
with open ('cameraMtx.pickle', "wb") as matrix_file:
    pickle.dump(cameraMtx, matrix_file, pickle.HIGHEST_PROTOCOL)
with open ('distCoeffs.pickle', "wb") as coeffs_file:
    pickle.dump(distCoeffs, coeffs_file, pickle.HIGHEST_PROTOCOL)
```

Luego en nuestro segundo programa, inicializamos un arreglo vacío llamado "correctedImg" para luego depositar en el mismo la información de la imagen corregida y deserializaremos los archivos pickle generados en el programa anterior haciendo el proceso inverso al anterior, cargando los archivos pickle en variables llamadas "cameraMtx" y "distCoeffs" respectivamente para su posterior uso.

```
correctedImg = []

with open ('cameraMtx.pickle', "rb") as cameraMtx_file:
    cameraMtx = pickle.load(cameraMtx_file)

with open ('distCoeffs.pickle', "rb") as distCoeffs_file:
    distCoeffs = pickle.load(distCoeffs_file)
```

Finalmente recorreremos las imágenes con un for igual al del programa anterior y aplicaremos la corrección a cada una de las imágenes con el método "undistort" de opencv, éste método recibirá la imagen en cuestión, la matriz de la cámara y los coeficientes de distorsión generados anteriormente, y la variable de salida. Como último paso se generan los archivos de imágenes en jpg.

```
for fname in images:
    img = cv2.imread(fname)
    correctedImg = cv2.undistort(img, cameraMtx, distCoeffs, correctedImg)
```

```
    cv2.imwrite('{:>2}.jpg'.format(i), correctedImg)
    i+=1
```

En la figura 2 podemos apreciar la imagen de la figura 1 con la corrección aplicada.

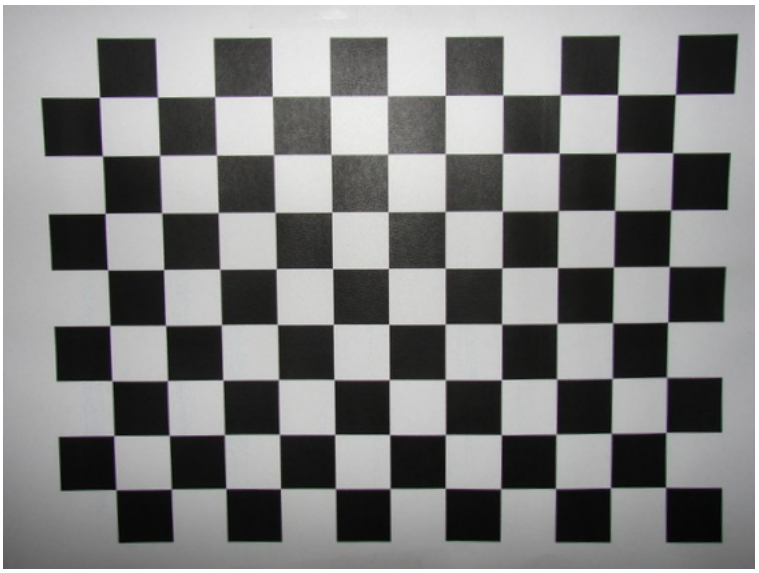


Figura 2: Imagen sin distorsión.

5. Conclusión

El método utilizado funciona muy bien si calibramos la misma para una sola distancia, de tener que calibrar la cámara para distintas distancias deberíamos repetir el proceso para cada una de estas.