Trabajo Práctico Final Fundamentos Matemáticos de la Visión Robótica (67.61) 2020

Román Vázquez Lareu, 100815

1/2/2020

Introducción

En el siguiente trabajo, se buscará desarrolar un software capaz de determinar la pose de ciertos objetos dispuestos en una zona de trabajo. Este proceso constará de 5 etapas.

- Calibración de los parámetros intrínsecos de la cámara: esto se hará a partir de un set de imágenes, donde se deberá justificar su elección.
- Calibración de los parámetros extrínsecos: esto se hará a partir de una unica imágen, logrando definir la terna de la zona de trabajo
- Desarrollo de algoritmo de detección de bloques: se buscará identificar su centro y vertices
- Desarrollo de método de validación del agloritmo generado
- Desarrollo de algoritmo de medición de bloques: se buscará identificar sus dimensiones

1 Calibración de parámetros Intrínsecos

Esta primera etapa se realizará haciendo uso de la función calibrateCamera() de openCV, basada en el algoritmo de Zhang. Tanto esta funcion como cualquier otra a usar en esta seccion y en la siguiente usan el modelo de pinhole. este modelo se caracteriza por formar la vista de una imagen proyectando los puntos 3D en el plano de la imagen usando la siguiente transformacion de perspectiva:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

Sin embargo, los lentes en general tienen cierta distorsión. En este caso particular, el lente de una camara web, tendrá un coeficiente k_1 de distorsión radial ($k_1 > 0$ será de barril, de lo contrario será de almohadon)

Previo al proceso de calibración conviene estudiar cada parámetro intrínseco con el objetivo de elegir correctamente el set de imagenes a usar basado en el resultado obtenido.

En primer lugar las distancias focales, f_x y f_y, se corresponden con la distancia entre el pinhole y el plano imagen. En el caso ideal de una camara pinhole, tendrán el mismo valor. En la práctica pueden diferir por varias razones: distorsión, errores en la calibración, etc. Sin embargo, su similitud será una propiedad a tener en cuenta al momento de elegir un set.

En segundo lugar el centro óptico, c_x y c_y, hacen referencia al punto en el plano imagen por donde pasa perpendicular al mismo y que atrviesa el pinhole. En un caso ideal, estará lo más cerca del centro posible.

En tercer lugar el skew, s, el cual tiene que ver con que el sensor no se encuentre perfectamente alineado al plano focal, pero en este caso será nulo para ambos sets de imágenes.

A continuación se presenta el código de calibración intrinseca. A lo largo de esta primera etapa, se indican los puntos del mundo real 3D que se querrán reconocer, con las distintas vistas del tablero de ajedrez, se utilizará la función de openCv *findChessCorners* para encontrar las coordenadas en pixeles de esos puntos, que se corresponderán con los puntos 3D provistos. Finalmente, con ambos set de puntos, se llama a la funcion *calibrateCamera* con el objetivo de obtener la matriz de parametros intrinsecos que relaciona ambos conjuntos de puntos y los coeficientes de distorsión.

[25]:
```
#Si queremos que las imágenes sean mostradas en una ventana emergente quitar el␣
 ↪inline
%matplotlib inline
# OpenCV-Python utiliza NumPy para el manejo de imágenes
import numpy as np
# cv2 es el módulo python para acceder a OpenCV
import cv2 as cv2
# Usamos las poderosas herramientas de graficación de matplotlib para mostrar␣
 ↪imágenes, perfiles, histogramas, etc
import matplotlib.pyplot as plt
```

[26]:
```
import glob
import PIL.ExifTags
import PIL.Image
```

[27]:
```
# Carpeta con las fotos:
calib_fnames = glob.glob('C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/
 ↪imagenes/img_cal_set1/*.png')

mostrar_figuras = True
```

[28]:
```
# Tamaño del tablero:
ch_size = (8, 6)

# lista de todos los puntos que vamos a recolectar
obj_points = list()
img_points = list()

# Lista de los puntos que vamos a reconocer en el mundo
# objp={(0,0,0), (1,0,0), (2,0,0) .... }
# corresponden a las coordenadas en el tablero de ajedrez.
objp = np.zeros((np.prod(ch_size), 3),  dtype=np.float32)
objp[:, :2] = np.mgrid[0:ch_size[0], 0:ch_size[1]].T.reshape(-1, 2)
objp = objp*28
```

[29]:
```
# Criterio de corte para el proceso iterativo de refinamientoi de esquinas.
# Parar si iteramos maxCount veces o si las esquinas se mueven menos de epsilon
maxCount = 30
epsilon = 0.001
criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_MAX_ITER, maxCount,␣
 ↪epsilon)
cb_flags = cv2.CALIB_CB_ADAPTIVE_THRESH
#cb_flags = cv2.CALIB_CB_FAST_CHECK

%matplotlib qt

for image_fname in calib_fnames:
    print("Procesando: " + image_fname , end='... ')
    img = cv2.imread(image_fname)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # para subpixel solamente␣
 ↪gray
    ret, corners = cv2.findChessboardCorners(img_gray, ch_size, flags=cb_flags)
    if ret:
        print('Encontramos esquinas!')
        obj_points.append(objp)
        print('Buscando esquinas en resolución subpixel', end='... ')
        corners_subp = cv2.cornerSubPix(img_gray, corners, (5, 5), (-1, -1),␣
 ↪criteria)
        print('OK!')
        img_points.append(corners_subp)
        cv2.drawChessboardCorners(img, ch_size, corners_subp, ret)
        if mostrar_figuras:
            plt.figure()
            plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
            plt.show()
```

```
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal10.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal11.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal12.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal13.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal14.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
```

```
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal15.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal16.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal17.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal18.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal19.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal2.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal20.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal3.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal4.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal5.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal6.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal7.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal8.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_cal_se
t1\img_cal9.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
```

[30]:
```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points,␣
↪img_gray.shape[::-1], None, None,flags=cv2.CALIB_ZERO_TANGENT_DIST)
```

```
#distCoeffs - Output vector of distortion coefficients (k1, k2, p1, p2[, k3[,␣
 ↪k4, k5, k6]])
#disotorsion tangencial p1=0,p2=0, puedo hacer cero a k2 y k3
dist[0][1]=0
dist[0][4]=0


print('Matriz parametros Intrinsecos:')
print('K=',mtx)
print('Distortion Coefficients : ')
print(dist)


%store mtx
%store dist
```

```
Matriz parametros Intrinsecos:
K= [[812.10141061   0.         316.58328056]
 [  0.         812.44382868 247.47338425]
 [  0.           0.           1.        ]]
Distortion Coefficients :
[[0.02314617 0.         0.         0.         0.        ]]
Stored 'mtx' (ndarray)
Stored 'dist' (ndarray)
```

[ ]:
```
Matriz parametros Intrinsecos set2:
K= [[[660.982170   0.         334.47300716]
 [  0.         663.73471966 229.39333938]
 [  0.           0.           1.        ]]
```

A simple vista se observa en primer lugar la ampliamente mayor similitud de las distanciasfocales correspondientes al set 1, respecto del set 2. Además, sabido el tamaño de las imá-genes (640x480 pixeles), el centro óptico del set 2 presenta un mayor offset, de esta maneraes conveniente elegir el set 1 de imágenes