Calibración de parámetros Extrínsecos

Esta etapa es muy similar a la anterior, pero lo que se busca es, por el mismo método, las coordenadas del mundo real de los puntos 3D usando una sola vista del patrón de ajedrez de tamaño conocido

[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

%store -r mtx
%store -r dist
```

[27]:
```
calib_fnames = glob.glob('C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/
 ↪imagenes/img_bloques_cal/imgCalExtr.png')
#calib_fnames = glob.glob('C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/
 ↪imagenes/img_bloques_desafio_cal/imgCalExtr.jpg')
```

[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 True:
            plt.figure()
            plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
            plt.show()
```

```
Procesando: C:/Users/pichichus/Desktop/Facultad/FMVR/TPFINAL/imagenes/img_bloque
s_cal/imgCalExtr.png... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!
```

[30]:
```
ret, rvecs, tvecs = cv2.solvePnP(obj_points[0], img_points[0],mtx,␣
 ↪dist,useExtrinsicGuess=False)


#Converts a rotation matrix to a rotation vector or vice versa
R = cv2.Rodrigues(rvecs)[0]


print('rotacion')
print('r=',R)
print('traslación')
print('t=',tvecs)
%store R
```

```
%store tvecs
```

```
rotacion
r= [[ 0.99922992 -0.0077504   0.03846435]
 [ 0.00790713  0.99996104 -0.00392445]
 [-0.03843243  0.00422557  0.99925227]]
traslación
t= [[-109.36482532]
 [-153.68304707]
 [ 702.90119809]]
Stored 'R' (ndarray)
Stored 'tvecs' (ndarray)
```

[ ]:

[ ]: