

TP9 Calibración de la Cámara

0. Resumen

1. Patrón
2. Sacar fotos desde diferentes puntos de vista
3. Encontrar esquinas
4. Ecuaciones de proyección patrón => foto

$$s \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = [K] [R_k | t_k] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \quad (1)$$

K: 5 (o 4) parámetros

R: 5 (o 9) parámetros

t: 3 parámetros

1. Hallar K, R_k, t_k y de yapa los coeficientes de distorsión.
2. Rectificar la imagen
3. Bonus: dibujar en 3D

1. Patrón

```
In [1]: import cv2
import numpy as np
import matplotlib.pyplot as plt
print("OpenCV version " + cv2.__version__)
print("Numpy version " + np.__version__)

# Importamos Librerías para manejo de tiempo
import time

%matplotlib inline
```

OpenCV version 4.0.1
Numpy version 1.19.2

```
In [2]: import glob

import PIL.ExifTags
import PIL.Image
```

2. Fotos desde distintos puntos de vista

```
In [8]: # Carpeta con las fotos
#calib_fnames = glob.glob('./otos prueba/*')
```

```

calib_fnames = glob.glob('./fotos/*')

print("Hay {} fotos del tablero".format(len(calib_fnames)))

mostrar_figuras = True

```

Hay 8 fotos del tablero

3, 4, 5. Identificación de Esquinas, Encontrar Matriz de Cámara

In [4]:

```

# Tamaño del tablero
ch_size = (9, 9)

# Listas 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)

#print(objp)

```

Vamos a encontrar las esquinas en el tablero con la función `findChessboardCorners` que tiene como parámetros a la imagen en escala de grises, el tamaño del tablero y un flag.

Existen distintos flags:

- `CALIB_CB_ADAPTIVE_THRESH`: Utiliza el umbral adaptativo para convertir la imagen a blanco y negro, en lugar de un nivel de umbral fijo.
- `CALIB_CB_FAST_CHECK`: Ejecuta una verificación rápida en la imagen que busca las esquinas del tablero de ajedrez y realiza un atajo en la llamada si no se encuentra ninguna. Esto puede acelerar el proceso.
- `CALIB_CB_NORMALIZE_IMAGE`: Normaliza la imagen con ecualización de histogramas antes de aplicar umbral fijo o adaptativo.
- `CALIB_CB_FILTER_QUADS`: Utiliza criterios adicionales (como el área del contorno o el perímetro) para filtrar los falsos quads extraídos en la etapa de recuperación del contorno.

In [5]:

```

# Criterio de corte para el proceso iterativo de refinamiento de esquinas.
# Parar si iteramos maxCount veces o si las esquinas se mueven menos de epsilon
maxCount = 25
epsilon = 0.001
criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_MAX_ITER, maxCount, epsilon)

#%%matplotlib qt

```

- Función para encontrar las esquinas cambiando el flag

In [6]:

```

def encontrar_esquinas(cb_flags):

    start = time.time()

    for image_fname in calib_fnames:

```

```

print("Procesando: " + image_fname, end='... ')
img = cv2.imread(image_fname)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
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)

    # Dibuja estos puntos que encontró
    cv2.drawChessboardCorners(img, ch_size, corners_subp, ret)

    if mostrar_figuras:
        plt.figure(figsize=(10,8))
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.show()
    else:
        print('No se encontraron esquinas')

elapsed = time.time() - start
print('\nTiempo de procesamiento {} segundos'.format(elapsed))

```

Con todos los flags fue posible encontrar las esquinas en el tablero. La diferencia entre ellos es el tiempo de ejecucion

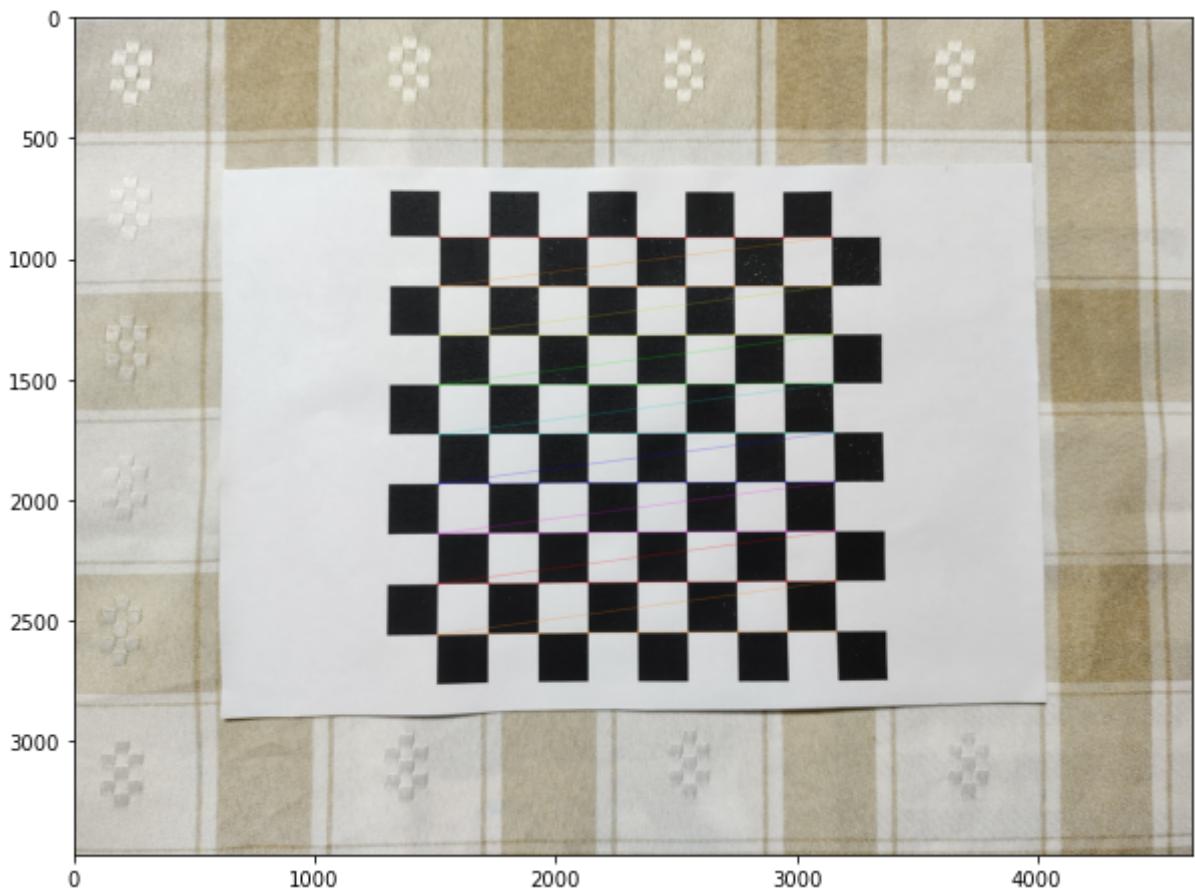
- CALIB_CB_ADAPTIVE_THRESH

Con este flag le lleva bastante tiempo a la funcion poder procesar todas las imagenes, por eso lo utilizamos con solo 4 fotos.

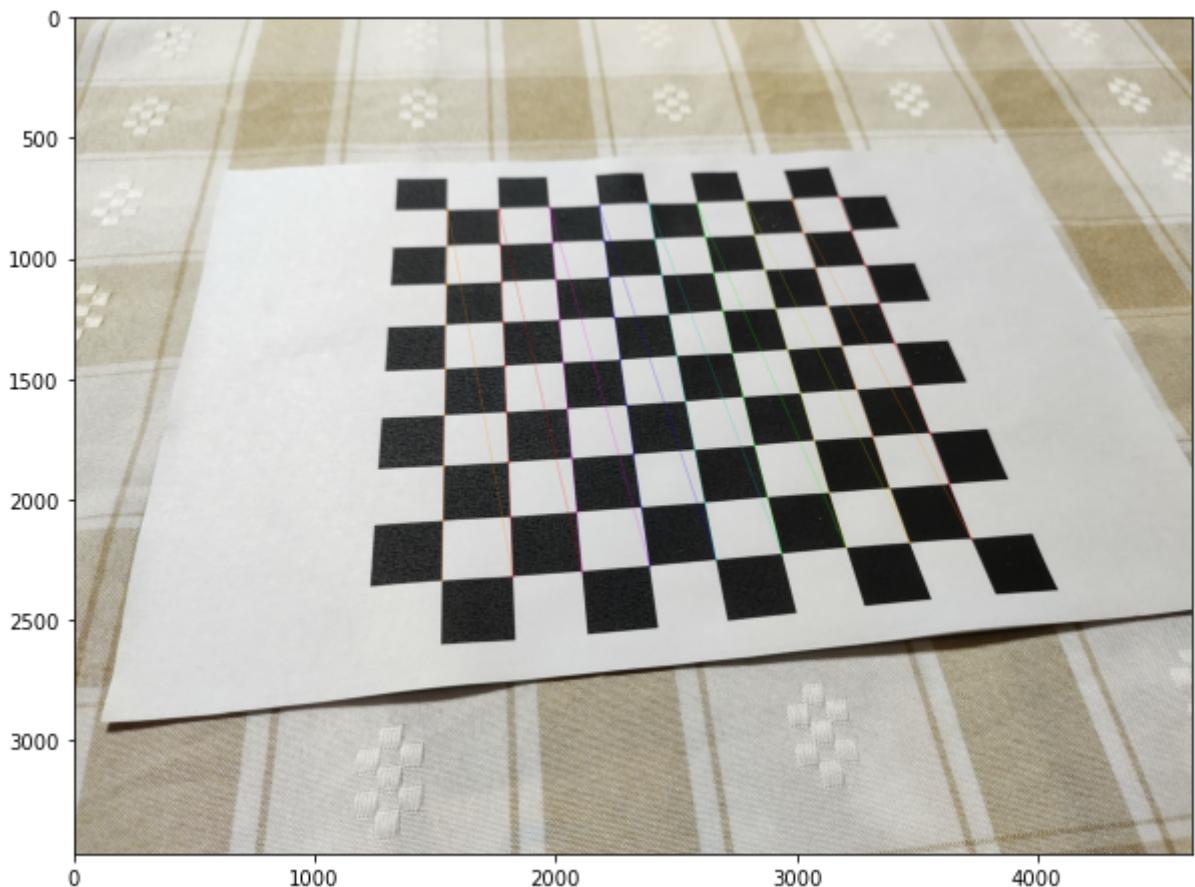
```
In [7]: obj_points, img_points = list(), list()

encontrar_esquinas(cv2.CALIB_CB_ADAPTIVE_THRESH)
```

Procesando: ./fotos prueba\Imagen (1).jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



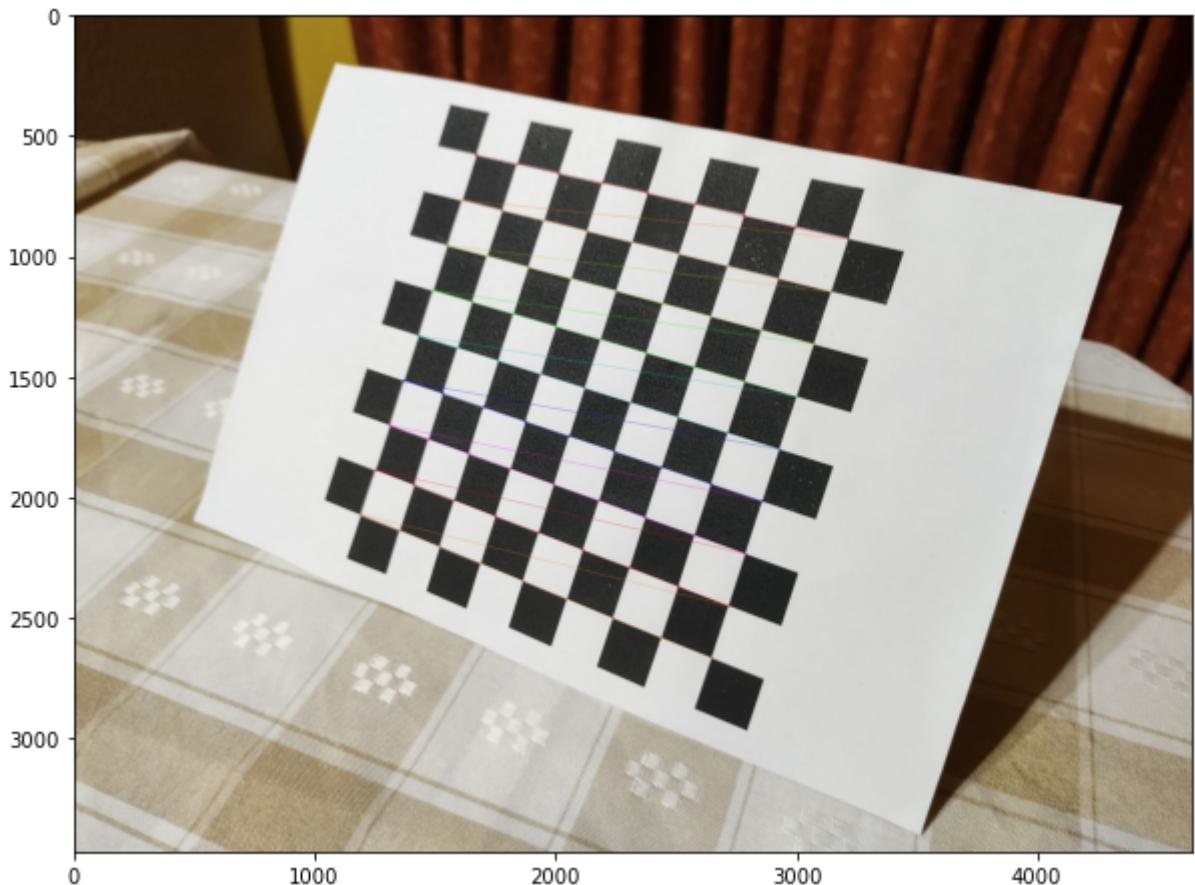
Procesando: ./fotos prueba\Imagen (2).jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos prueba\Imagen (3).jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos prueba\Imagen (4).jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Tiempo de procesamiento 117.59746813774109 segundos

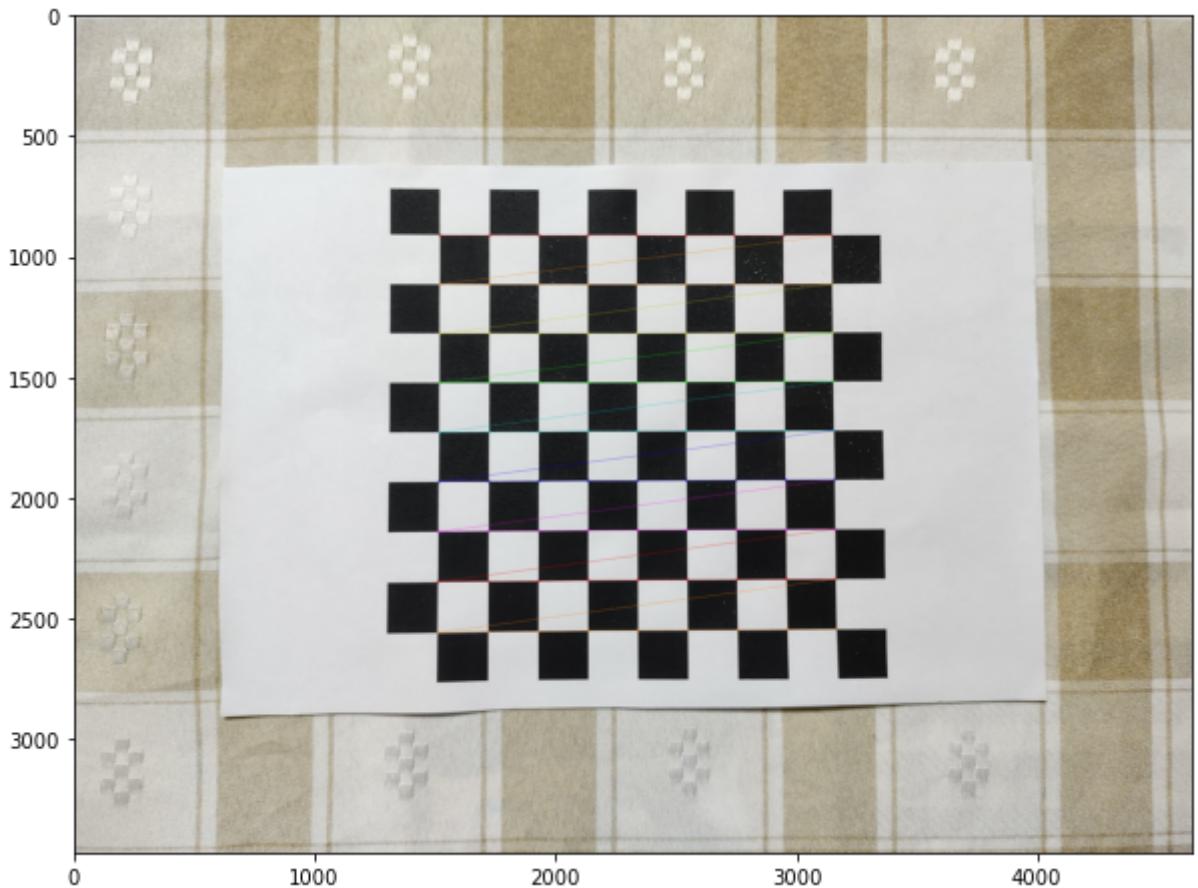
Los siguientes flags pudimos evaluarlos sin ningun problema en las 8 fotos sacadas.

- CALIB_CB_FILTER_QUADS

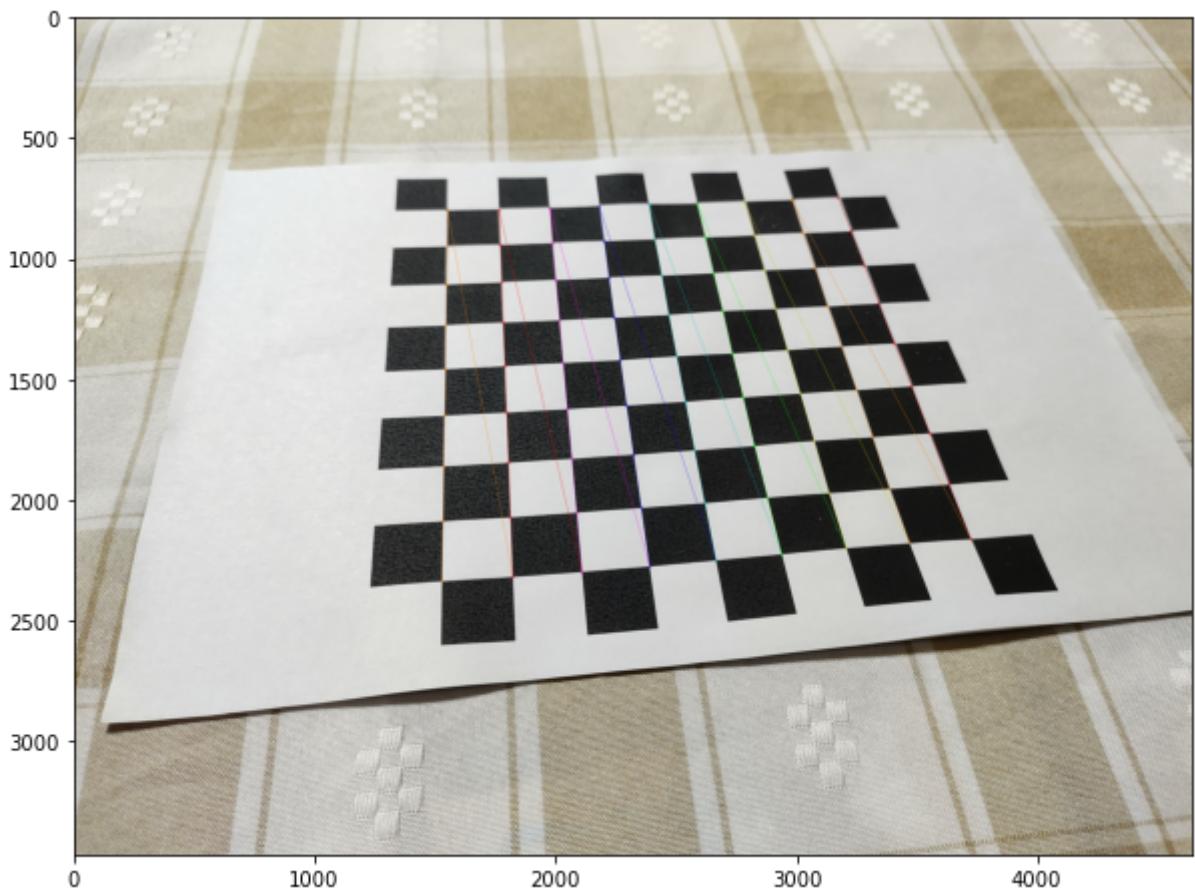
Este es el flag donde el proceso es el mas rapido.

```
In [9]:  
    obj_points, img_points = list(), list()  
  
    encontrar_esquinas(cv2.CALIB_CB_FILTER_QUADS)
```

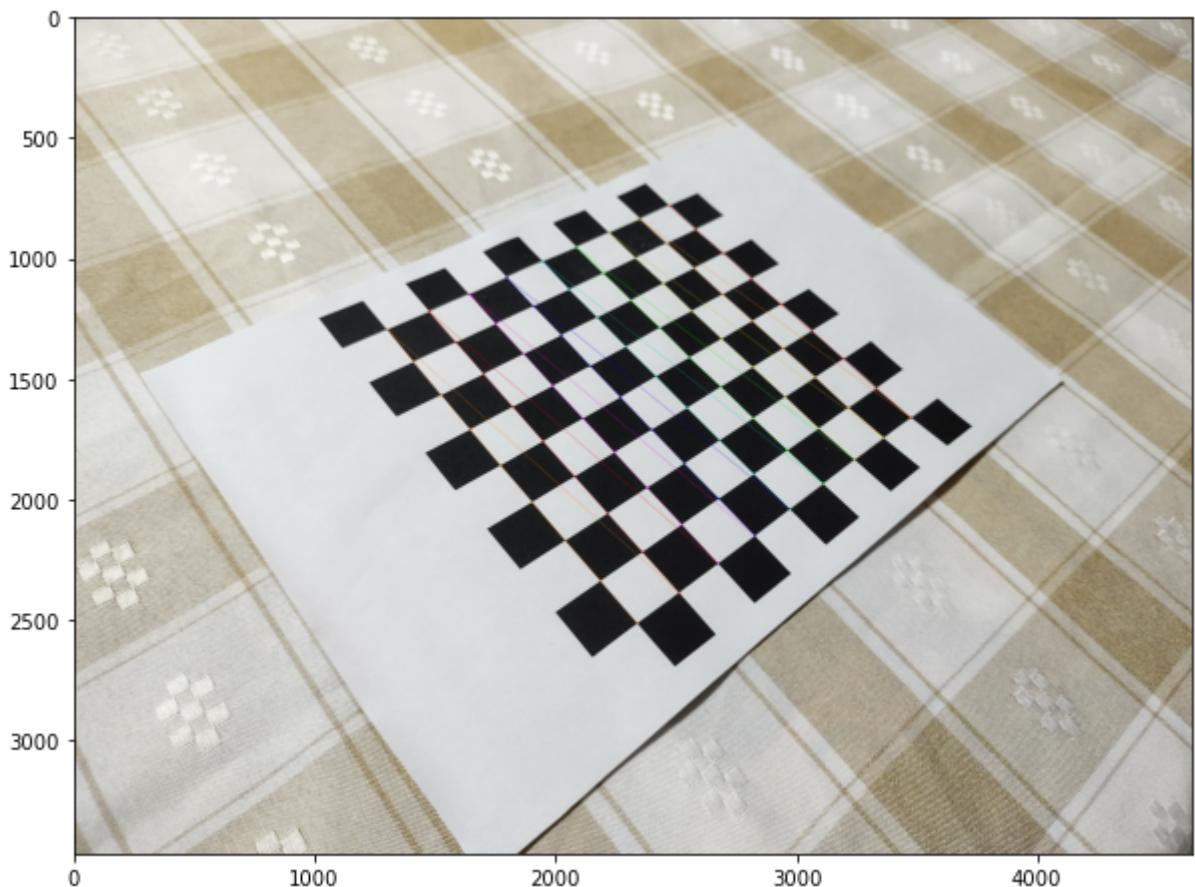
Procesando: ./fotos\Imagen 01.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



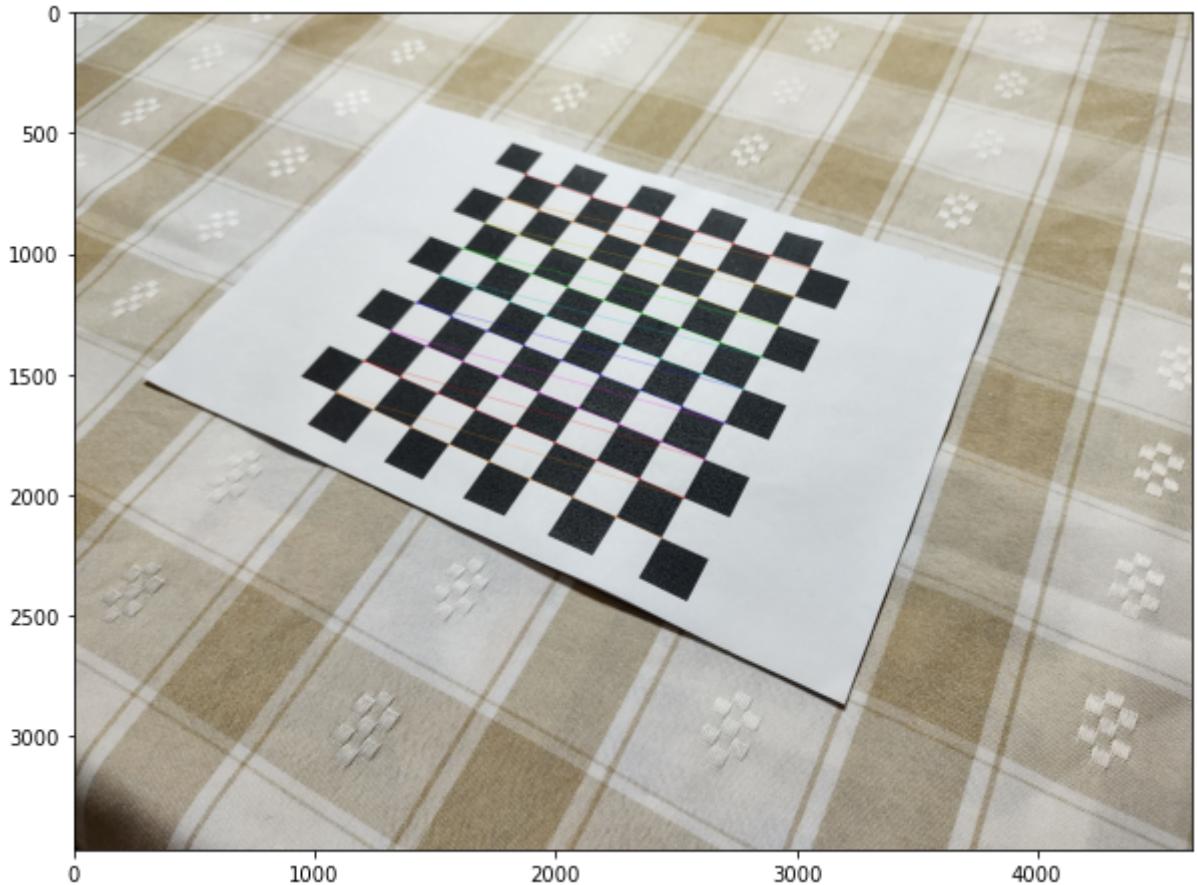
Procesando: ./fotos\Imagen 02.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



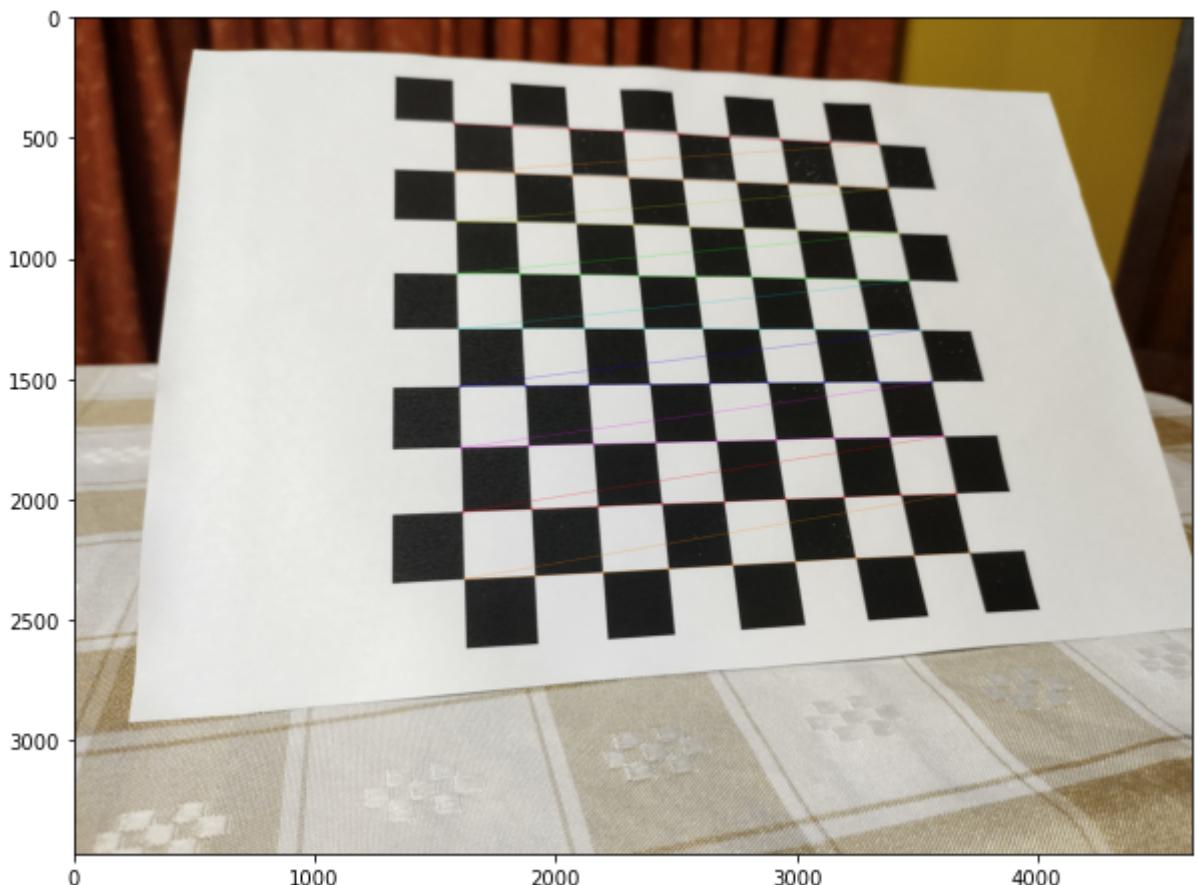
Procesando: ./fotos\Imagen 03.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



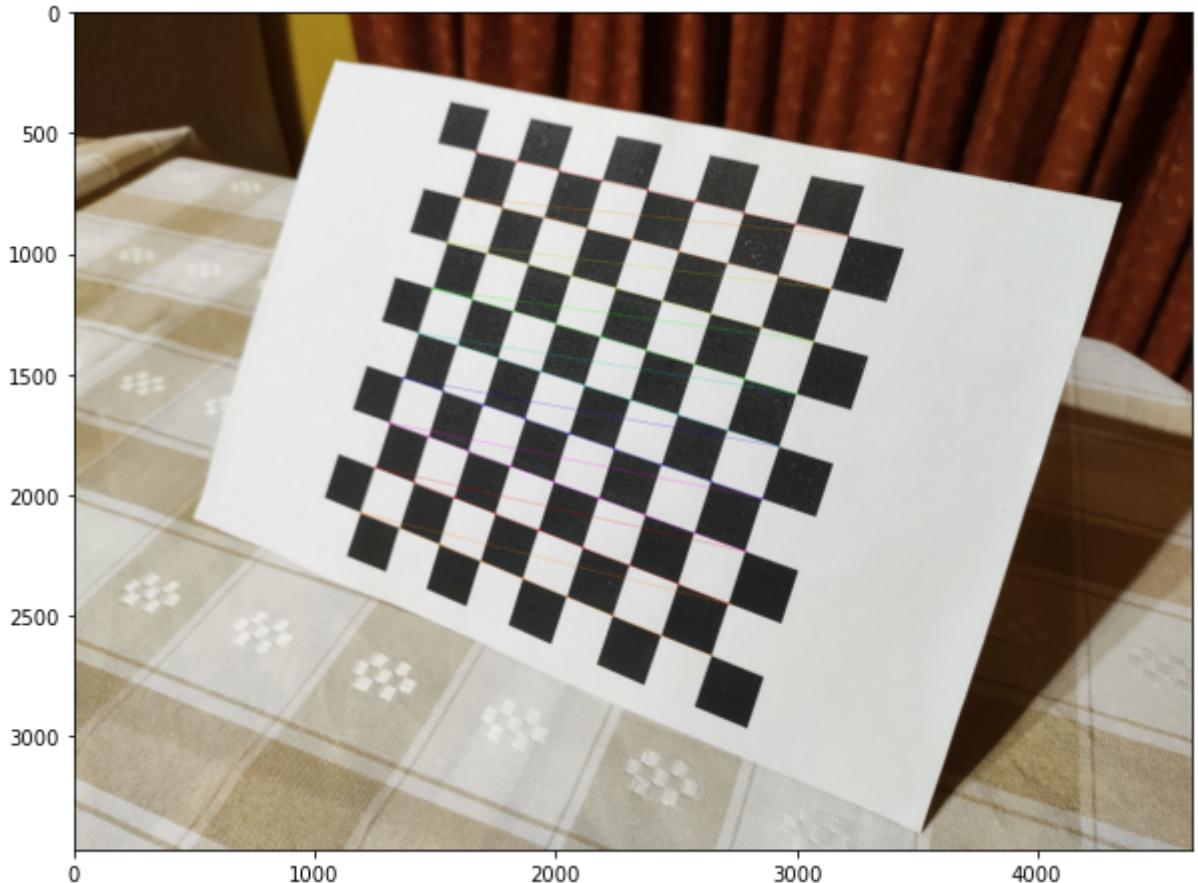
Procesando: ./fotos\Imagen 04.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 05.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 06.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 07.jpg... No se encontraron esquinas
Procesando: ./fotos\Imagen 08.jpg... No se encontraron esquinas

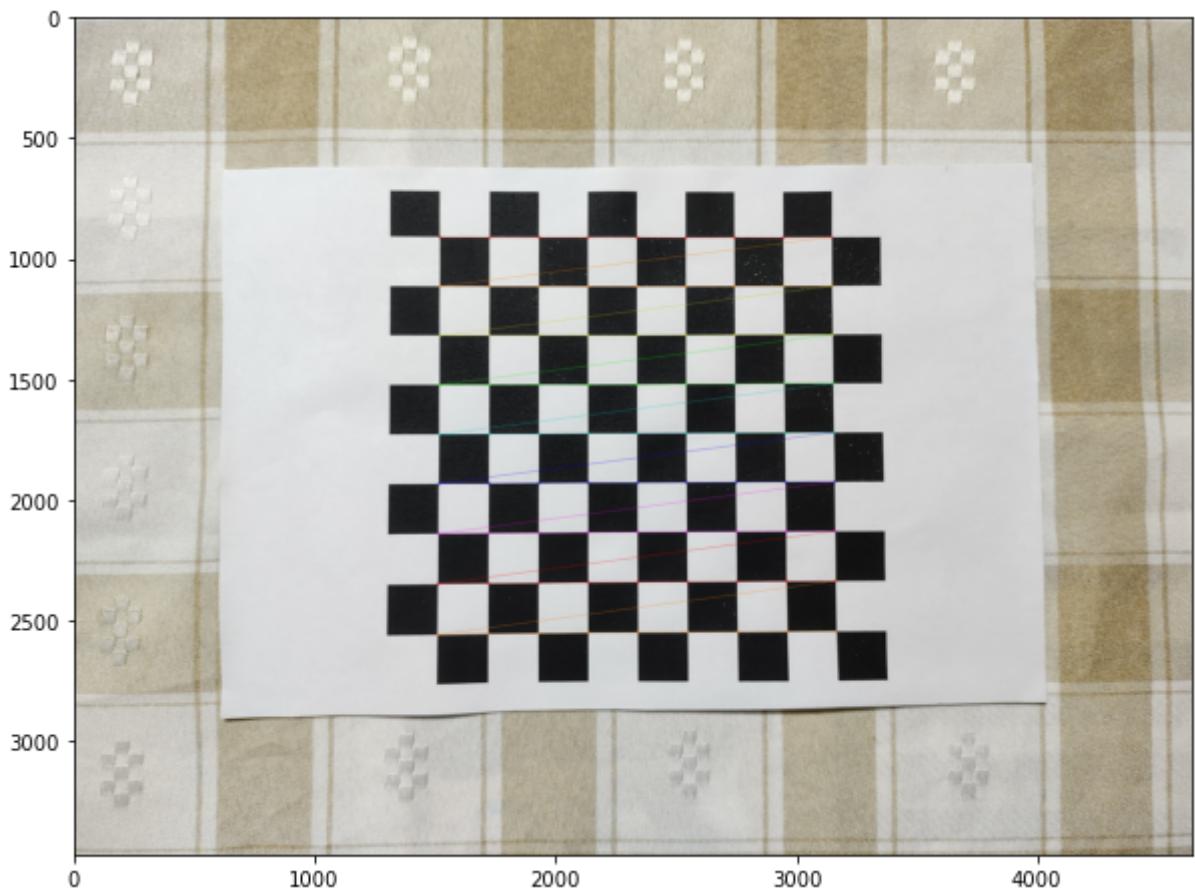
Tiempo de procesamiento 41.24717831611633 segundos

- CALIB_CB_FAST_CHECK

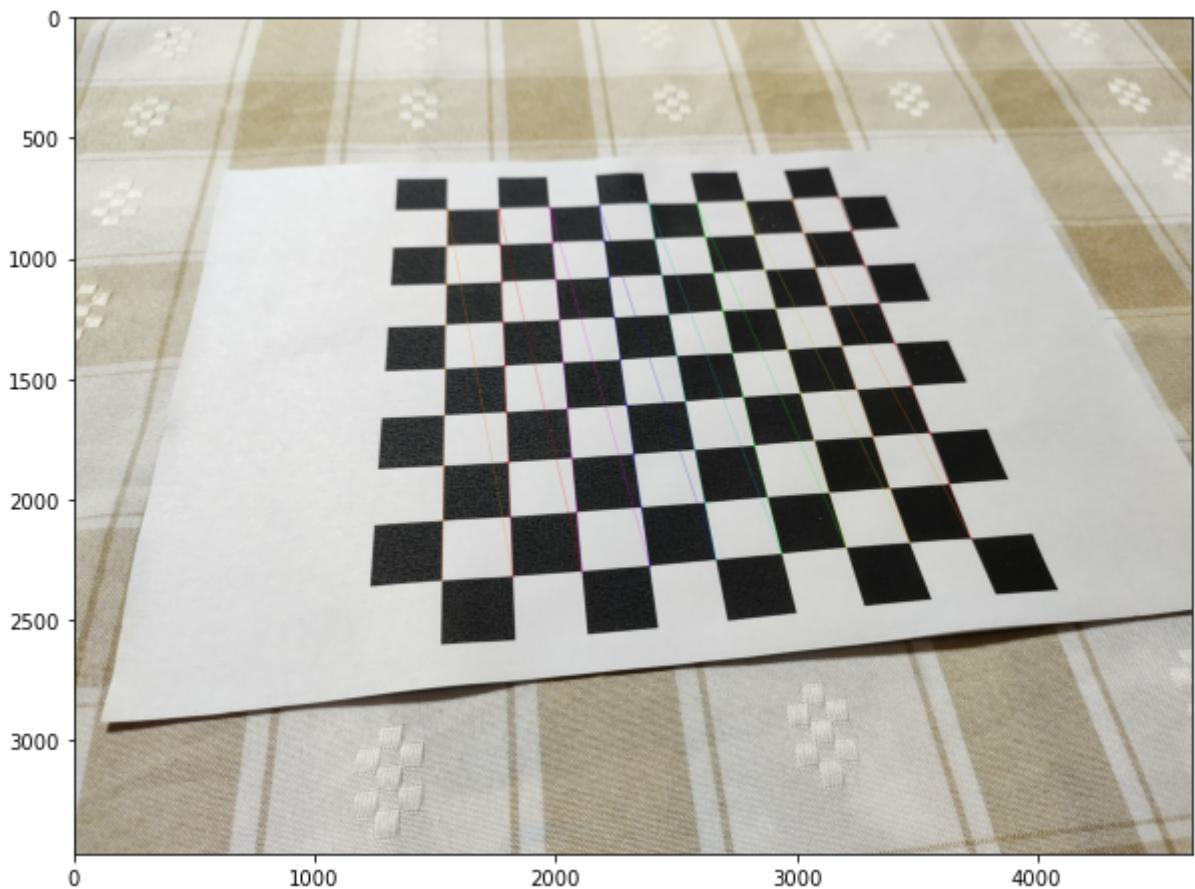
El tiempo de procesamiento es mas rapido que CALIB_CB_ADAPTIVE_THRESH, pero mas lento que CALIB_CB_FILTER_QUADS.

```
In [10]: obj_points, img_points = list(), list()  
encontrar_esquinas(cv2.CALIB_CB_FAST_CHECK)
```

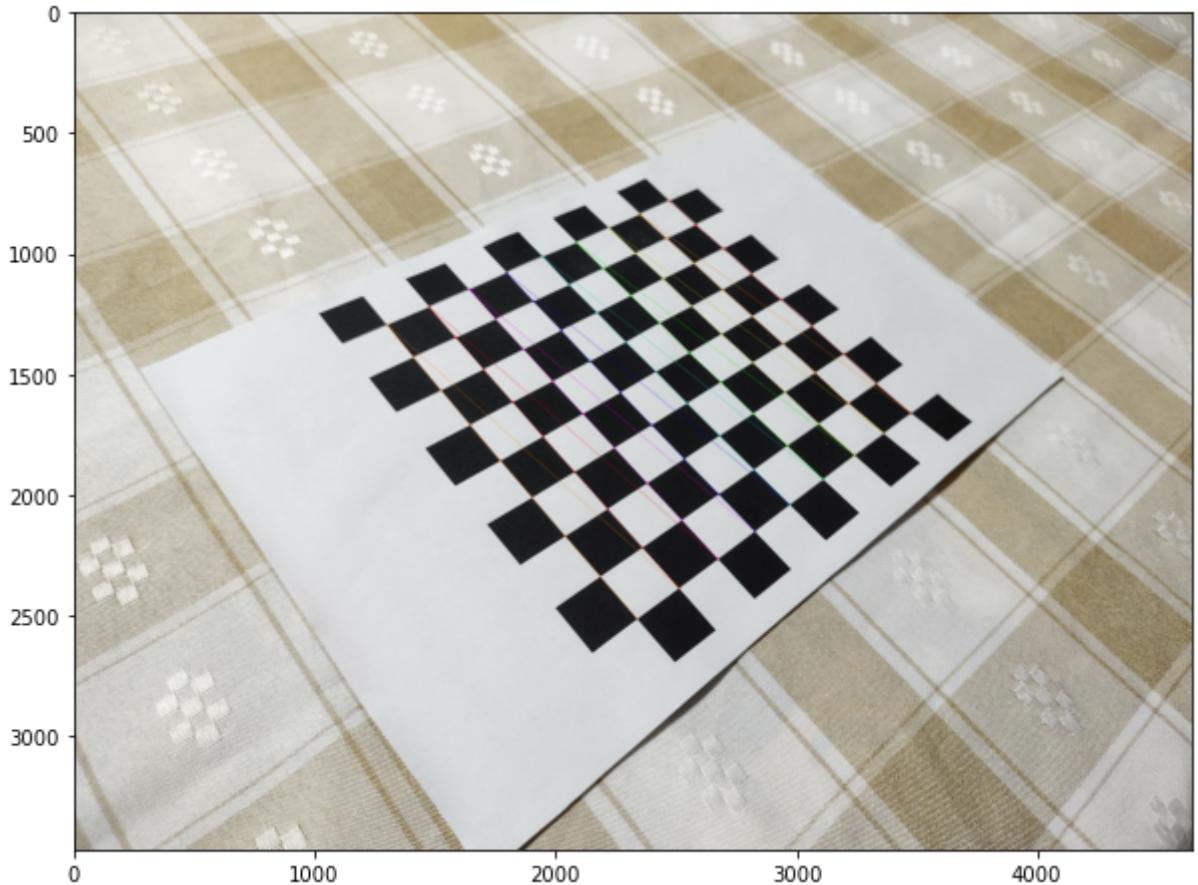
Procesando: ./fotos\Imagen 01.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



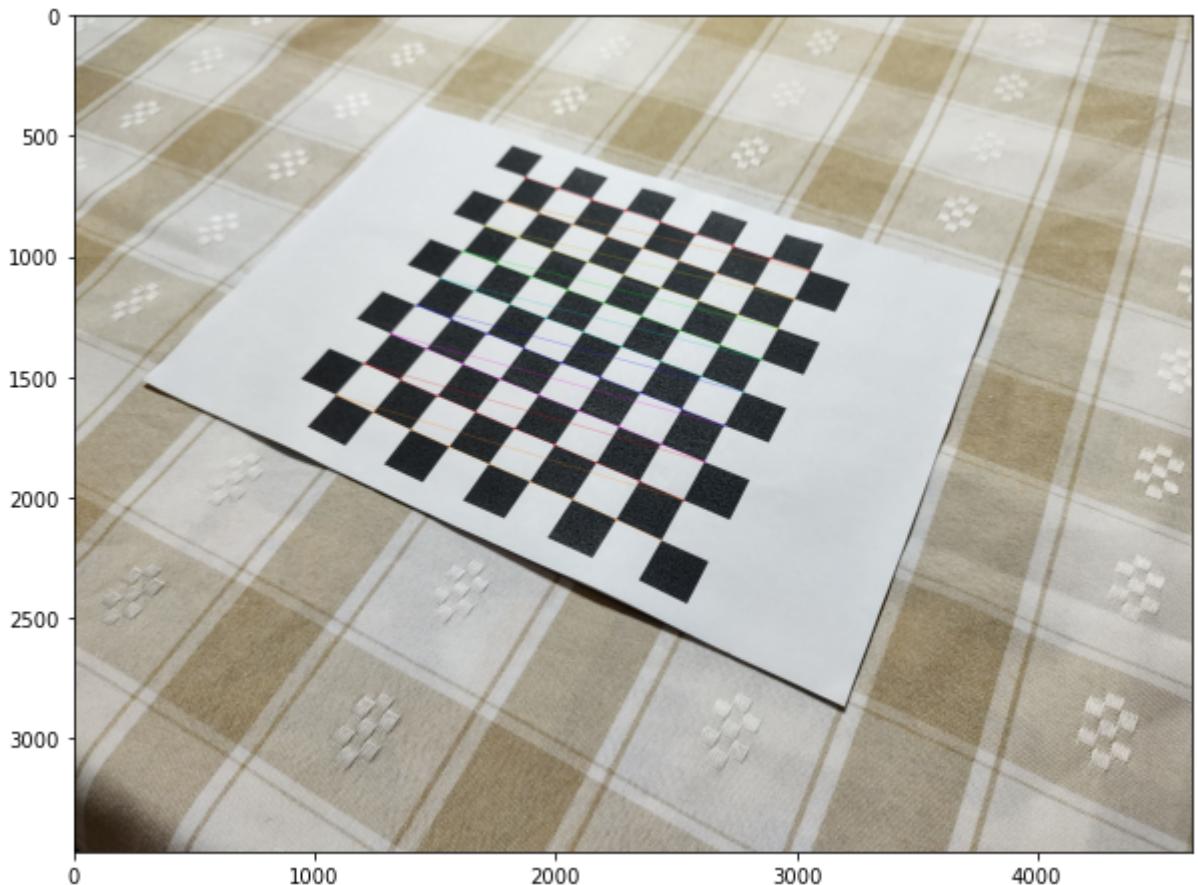
Procesando: ./fotos\Imagen 02.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



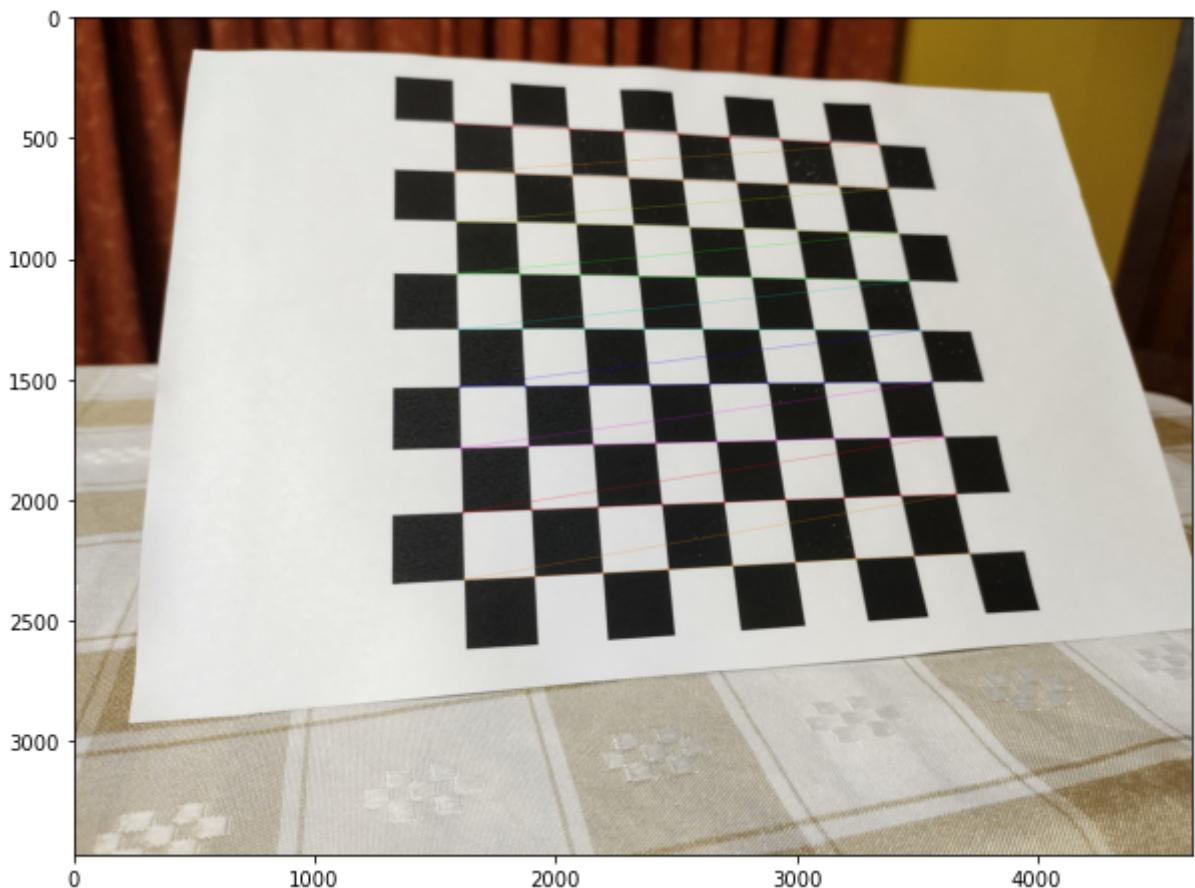
Procesando: ./fotos\Imagen 03.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



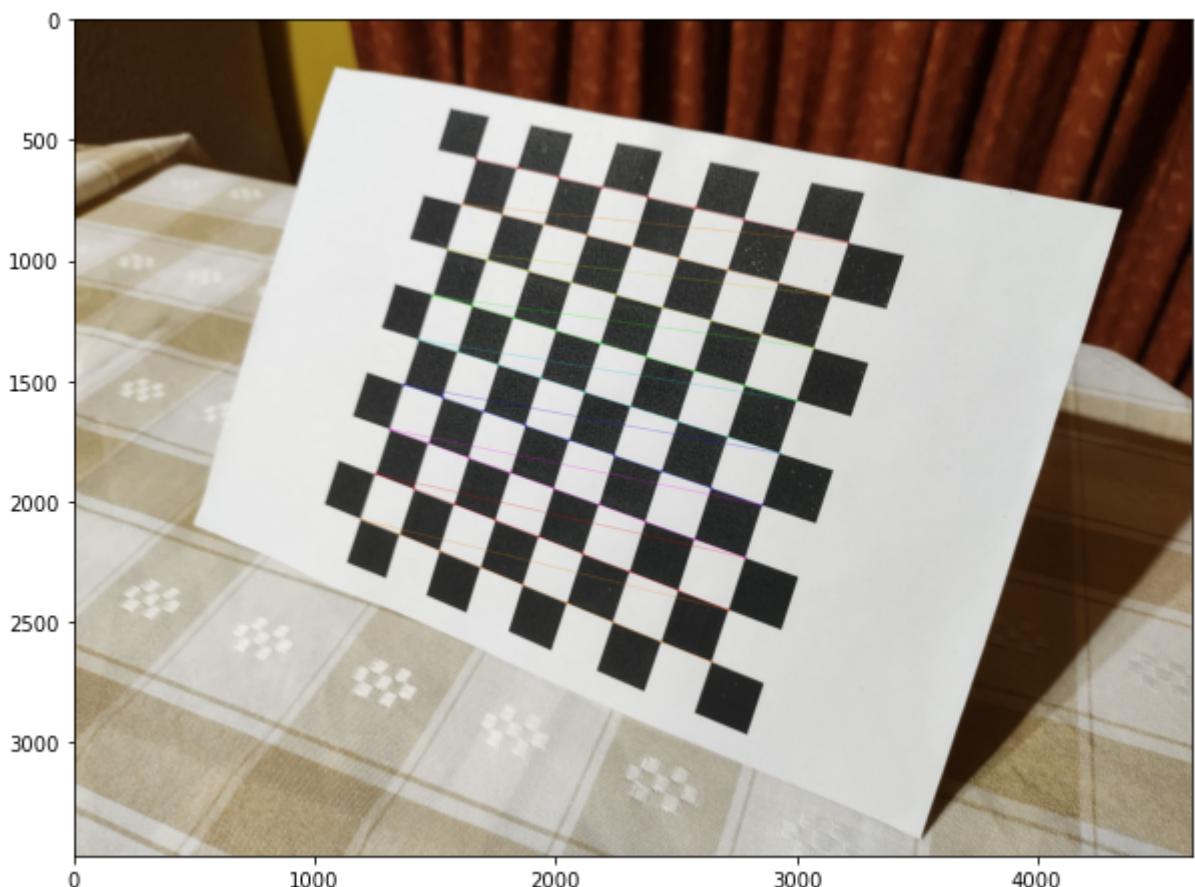
Procesando: ./fotos\Imagen 04.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 05.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 06.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 07.jpg... No se encontraron esquinas
Procesando: ./fotos\Imagen 08.jpg... No se encontraron esquinas

Tiempo de procesamiento 91.57355904579163 segundos

```
In [11]: print("Cantidad de puntos imagen: ", len(img_points))
```

Cantidad de puntos imagen: 6

Con los dos ultimos flags se pudieron encontrar las esquinas en 6 de las 8 fotos sacadas.

5. Calibración

Utilizamos la funcion calibrateCamera que tiene como parametros los puntos objeto, los puntos imagen y el tamaño de la imagen.

Esta funcion devuelve los siguientes valores:

- ret: boolean si funcionó o no
- mtx: matriz de la camara (matriz de parametros intrinsecos)
- dist: coeficientes de distorsion
- rvecs: vectores de rotacion
- tvecs: vectores de traslacion

La matriz de la camara es:

$$K = \begin{bmatrix} f_x & 0 & t_x \\ 0 & f_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

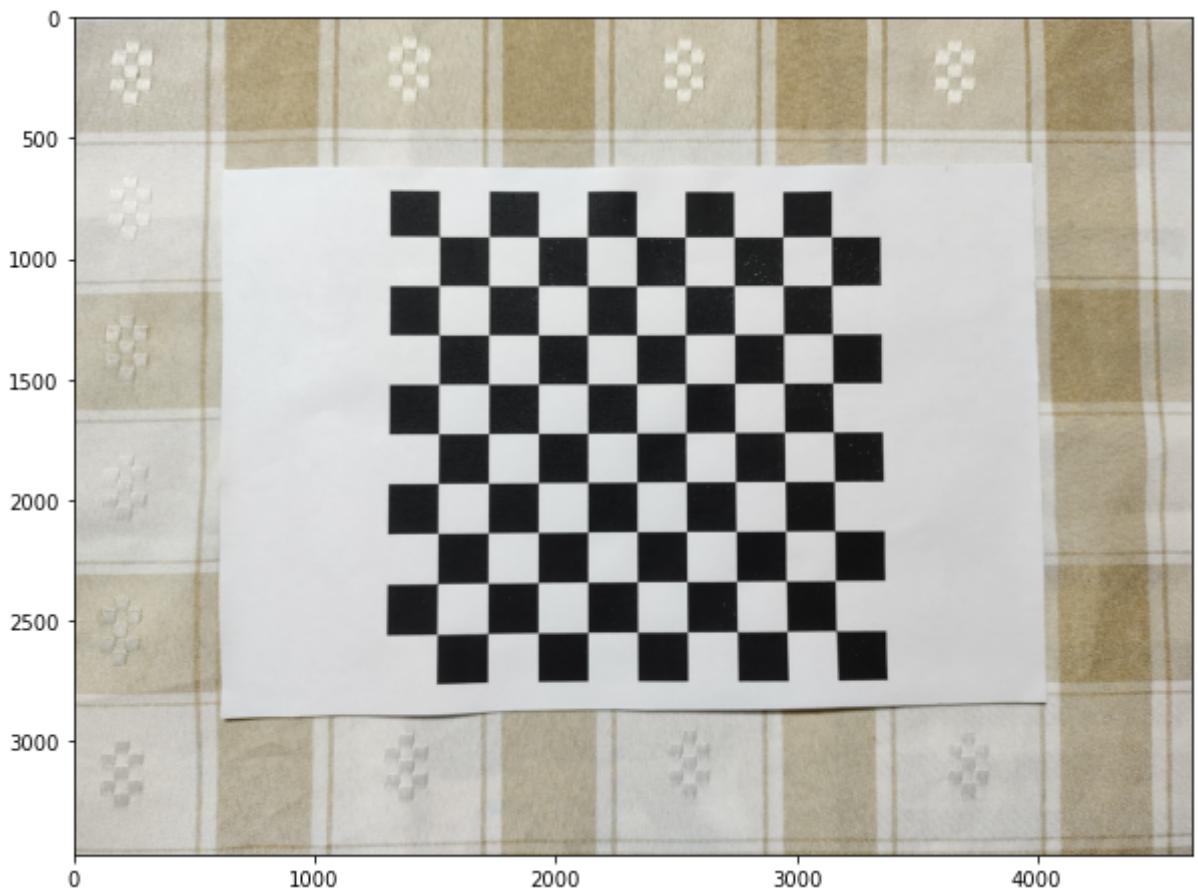
In [12]:

```
idx=0

img = cv2.imread(calib_fnames[idx])
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

h, w = img_gray.shape

plt.figure(figsize=(10,8))
plt.imshow(imgRGB)
plt.show()
```



```
In [13]: ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points, (w, h), N

print('Matriz de la camara = ')
print(mtx)

print('\nCoeficientes de distorsion = ')
print(dist)
```

```
Matriz de la camara =
[[3.46785470e+03 0.00000000e+00 2.34241813e+03]
 [0.00000000e+00 3.46700670e+03 1.72400515e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

```
Coeficientes de distorsion =
[[ 1.17215277e-01 -1.11188740e+00 -3.44442680e-03 7.36021141e-04
  3.13644405e+00]]
```

```
In [14]: print('Vectores de rotacion = ')
print(rvecs)

print('\nVectores de traslacion = ')
print(tvecs)
```

```
Vectores de rotacion =
[array([[ -0.0455273 ,
   -0.01865399],
  [-0.00379824]]), array([[ -0.54608942],
 [ 0.40463852],
 [ 1.41775656]]), array([[ -0.66329193],
 [ 0.12041721],
 [ 0.87964818]]), array([[ -0.57660089],
 [ 0.17743824],
 [ 0.45057735]]), array([[ -0.35697143],
 [-0.19630638],
 [-0.057122 ]]), array([[ -0.04470377],
```

```

[ 0.3218839 ],
[ 0.31134557]]]

Vectores de translacion =
[array([[ -4.03924832],
       [-3.97798156],
       [17.10839301]]), array([[ 4.16427854],
       [-4.86300779],
       [17.49981653]]), array([[ 0.78140044],
       [-5.71190959],
       [21.21232907]]), array([[-2.99768345],
       [-6.70587346],
       [22.4444875 ]]), array([[-3.20396278],
       [-5.38970082],
       [14.65502428]]), array([[-3.42444817],
       [-5.81954251],
       [17.75411749]])]

```

In [15]:

```

#En la foto se guarda la distancia focal
print('Leyendo datos del header EXIF ...')
exif_img = PIL.Image.open(calib_fnames[idx])

exif_data = {
    PIL.ExifTags.TAGS[k]: v
    for k, v in exif_img._getexif().items()
    if k in PIL.ExifTags.TAGS
}

# Si quiero ver toda la informacion de la imagen:
#print('Full exif dump:')
#import pprint
#pprint.pprint(exif_data)

focal_length_exif = exif_data['FocalLength']
print('Distancia focal = ', focal_length_exif, 'mm')

```

Leyendo datos del header EXIF ...

Distancia focal = 5.53 mm

Todas las fotos tienen una distancia focal de 5.53 mm, por lo tanto no coincide con el valor obtenido en la matriz de la camara, tanto en la distancia focal en x como en y.

6. Rectificación (undistort)

Rectificamos todas las imagenes donde se encontraron esquinas. Quitamos la distorsion que tienen con la funcion *undistort* que toma como parametros a la imagen, la matriz intrinseca de la camara y los coeficientes de distorsion.

In [16]:

```

img_dst = []

for i in range(6):
    print('Rectificando imagen {}...'.format(i+1))
    img_to_undistort = cv2.imread(calib_fnames[i])

    dst = cv2.undistort(img_to_undistort, mtx, dist)
    dst_rgb = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)

    #Guardamos las imagenes rectificadas en una lista
    img_dst.append(dst)

plt.figure(figsize=(10,8))
plt.imshow(dst_rgb)
plt.show()

```

Rectificando imagen 1...



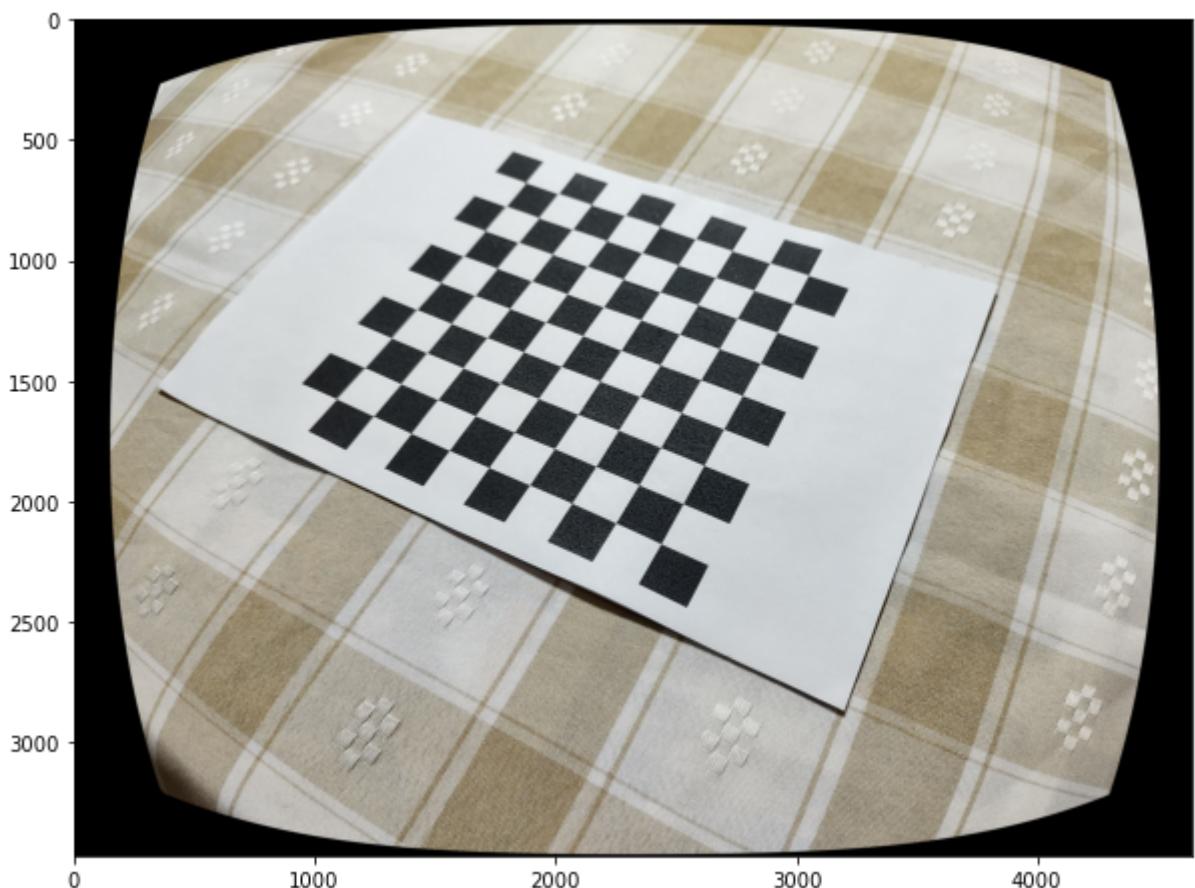
Rectificando imagen 2...



Rectificando imagen 3...



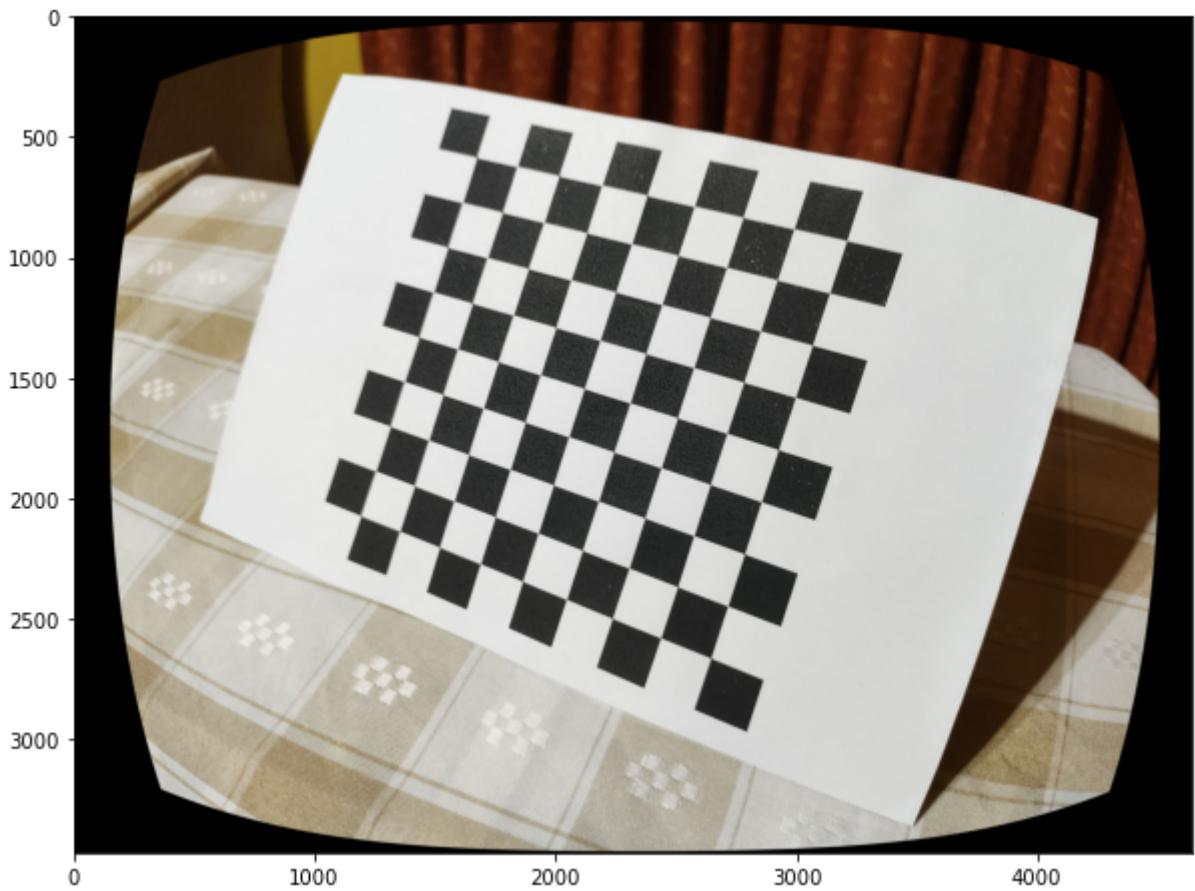
Rectificando imagen 4...



Rectificando imagen 5...



Rectificando imagen 6...



A. Bonus: Dibujando en el Espacio

Dibujamos en las imágenes donde se encontraron esquinas.

```
In [17]: # Vertices 3x3
esquinas1 = np.float32([[0,0,0], [0,3,0], [3,3,0], [3,0,0],
[0,0,-3],[0,3,-3],[3,3,-3],[3,0,-3]])

# Vertices 1x1
esquinas2 = np.float32([[0,0,0], [0,1,0], [1,1,0], [1,0,0],
[0,0,-1],[0,1,-1],[1,1,-1],[1,0,-1]])
```

Dibujando un eje

```
In [18]: def dibujar_eje(idx, esq_3d):

    esquinas_2d, _ = cv2.projectPoints(esq_3d, rvecs[idx], tvecs[idx], mtx, dist)
    esq_2d = esquinas_2d[:,0, :]

    r = (0, 0, 255) # r (in BGR)
    b = (255, 0, 0) # b (in BGR)
    g = (0, 255, 0) # g (in BGR)
    line_width = 25

    img=cv2.imread(calib_fnames[idx])

    #Redondeamos los valores de los puntos porque cv.line no toma puntos en np.float
    esq_round = []
    for esquina in esq_2d:
        x = round(esquina[0])
        y = round(esquina[1])

        round_points = (x,y)
        esq_round.append(round_points)

    # Eje x
    cv2.line(img, esq_round[0], esq_round[1], r, line_width)

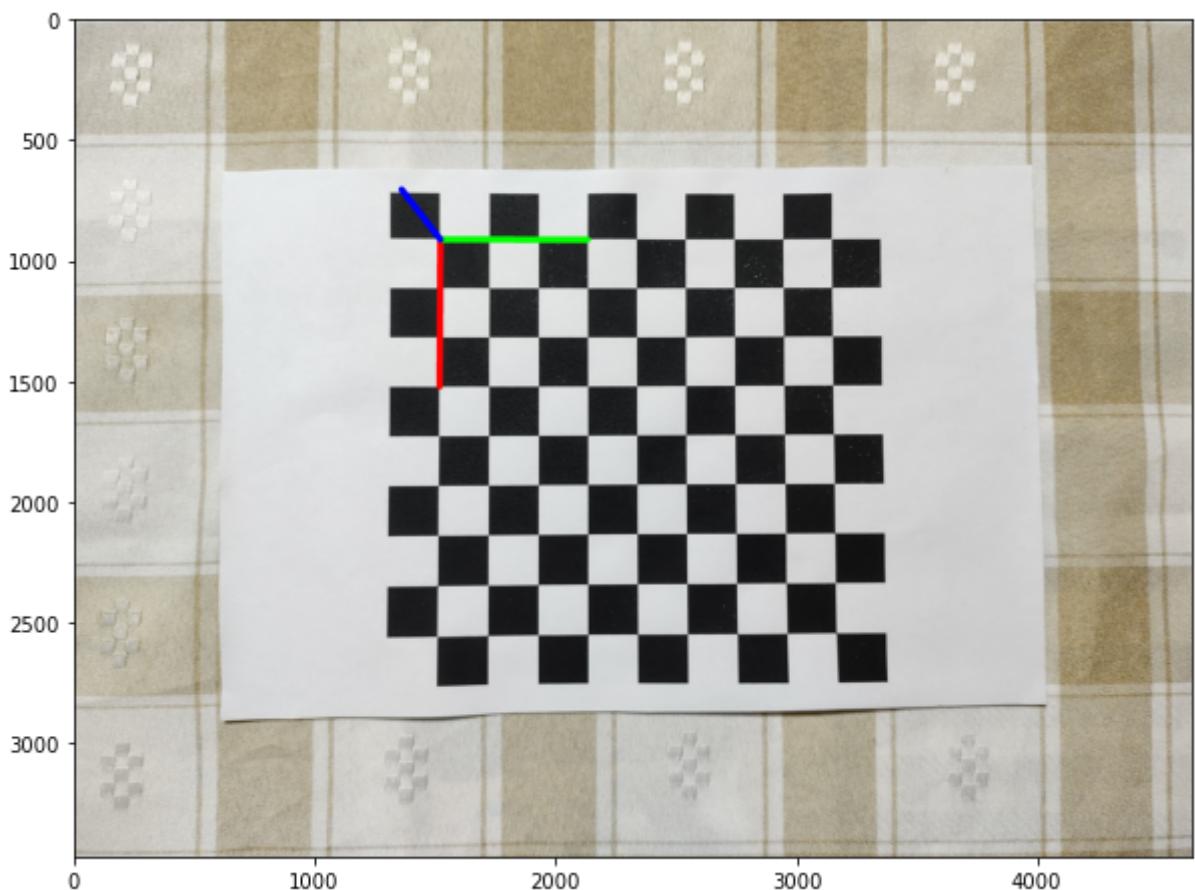
    # Eje y
    cv2.line(img, esq_round[0], esq_round[3], g, line_width)

    # Eje z
    cv2.line(img, esq_round[0], esq_round[4], b, line_width)

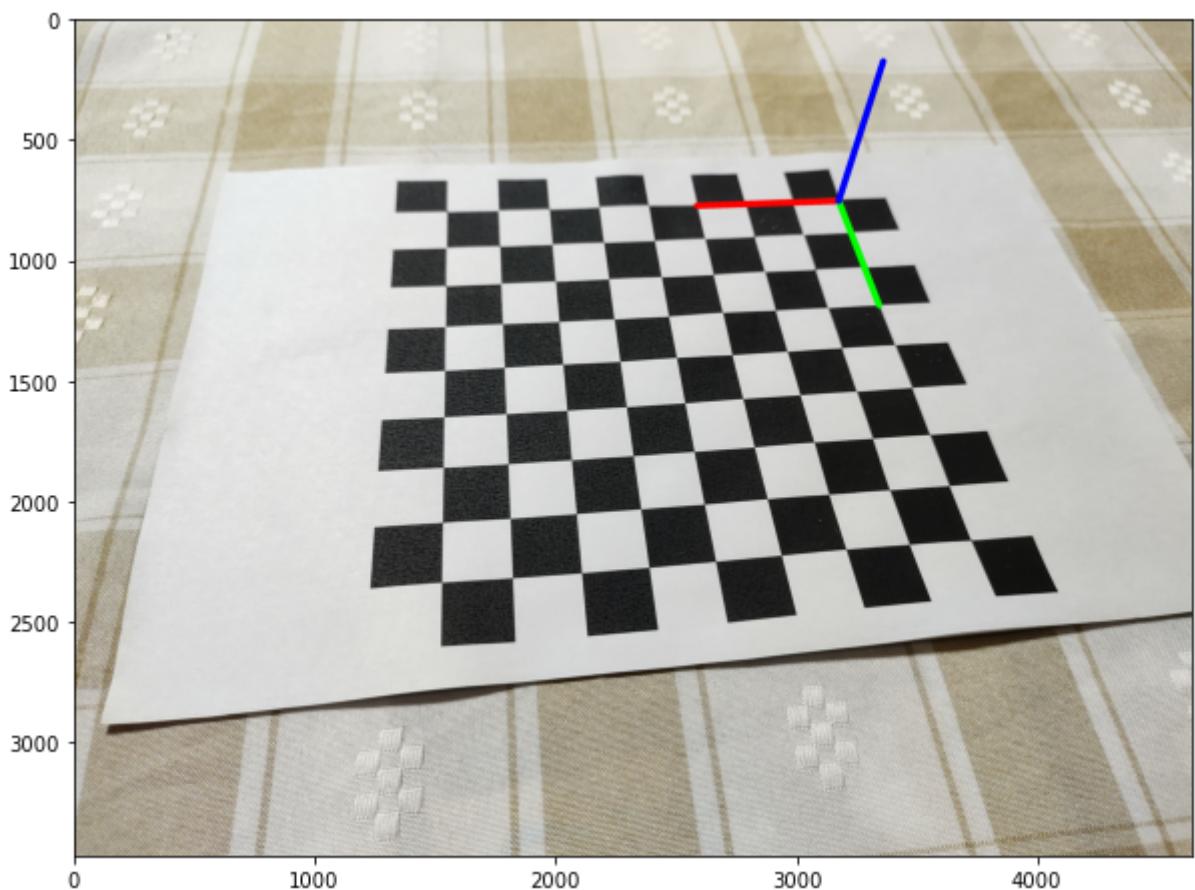
    plt.figure(figsize=(10,8))
    plt.imshow(img[...,:-1])
    plt.show()
```

```
In [20]: for i in range(6):
    print("Dibujando en la imagen ", i+1)
    dibujar_eje(i,esquinas1)
```

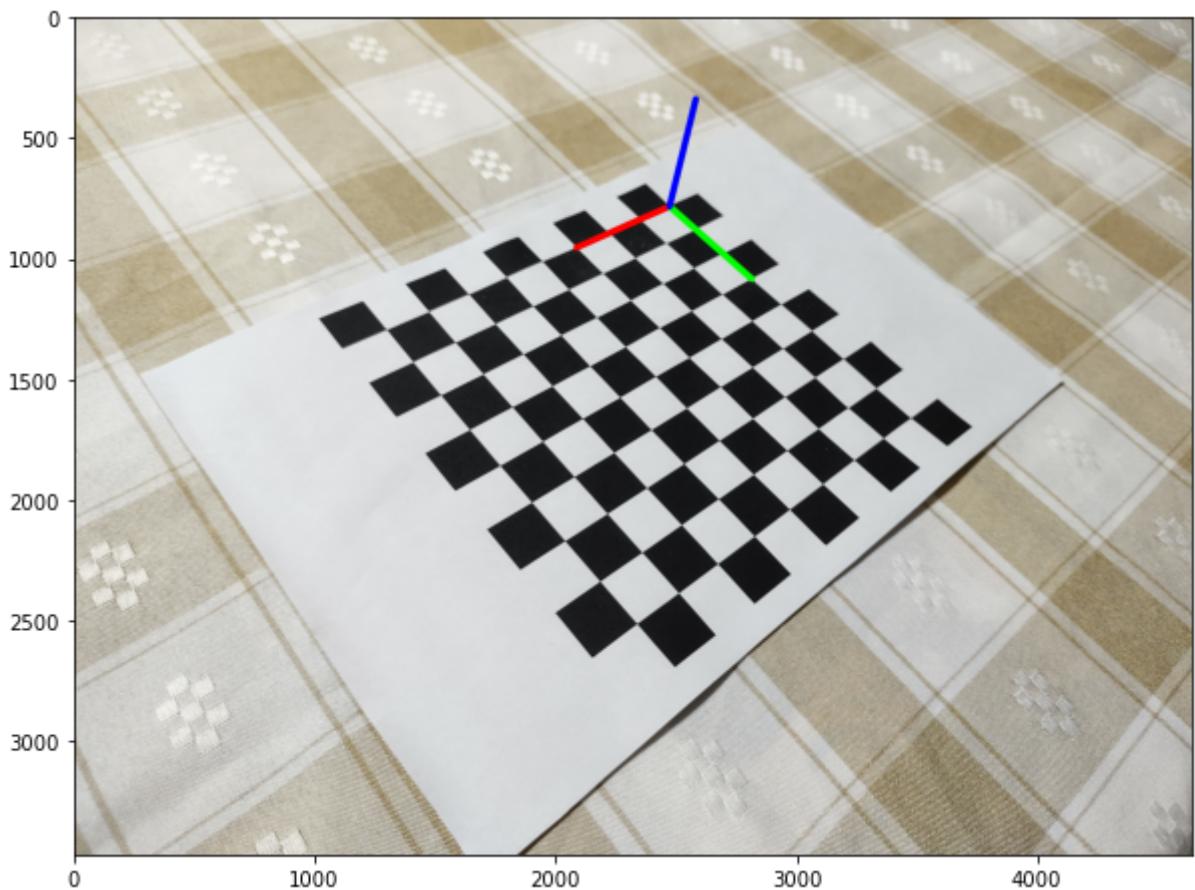
Dibujando en la imagen 1



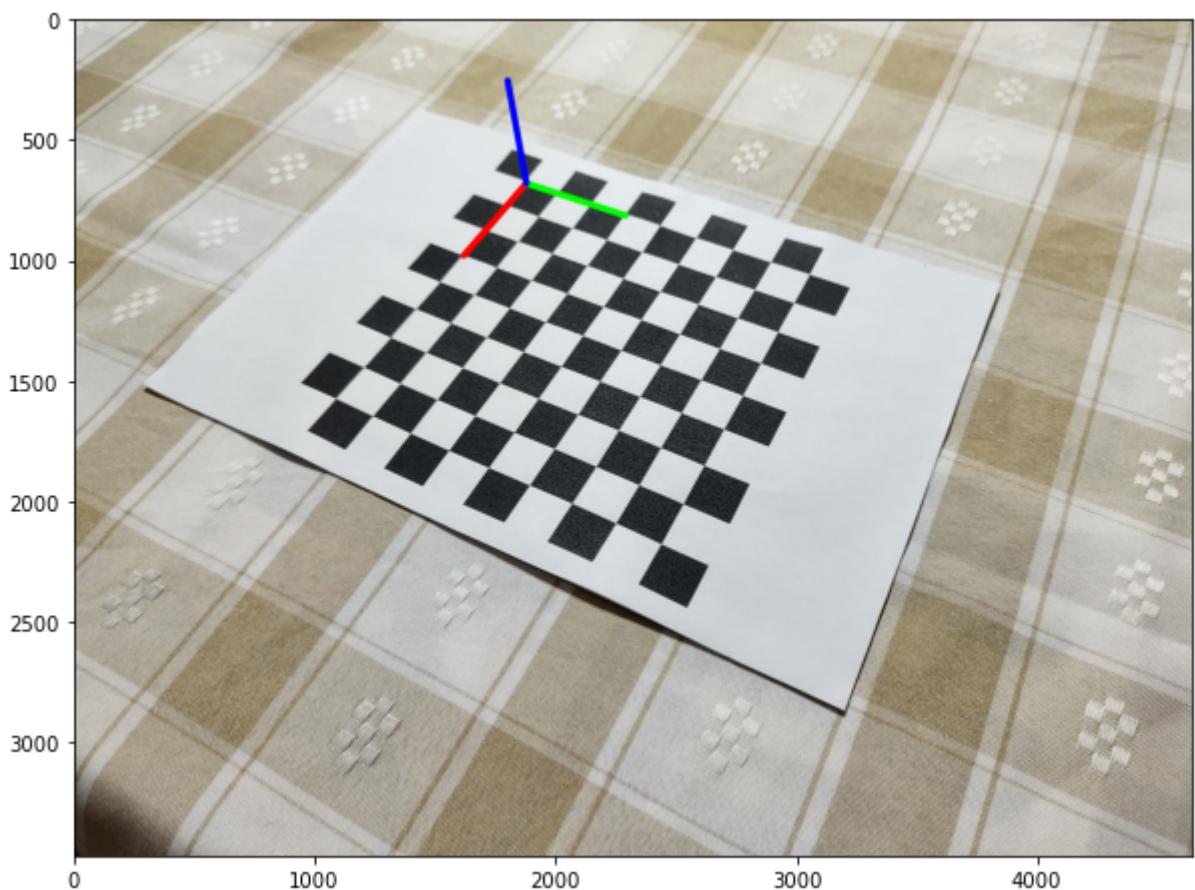
Dibujando en la imagen 2



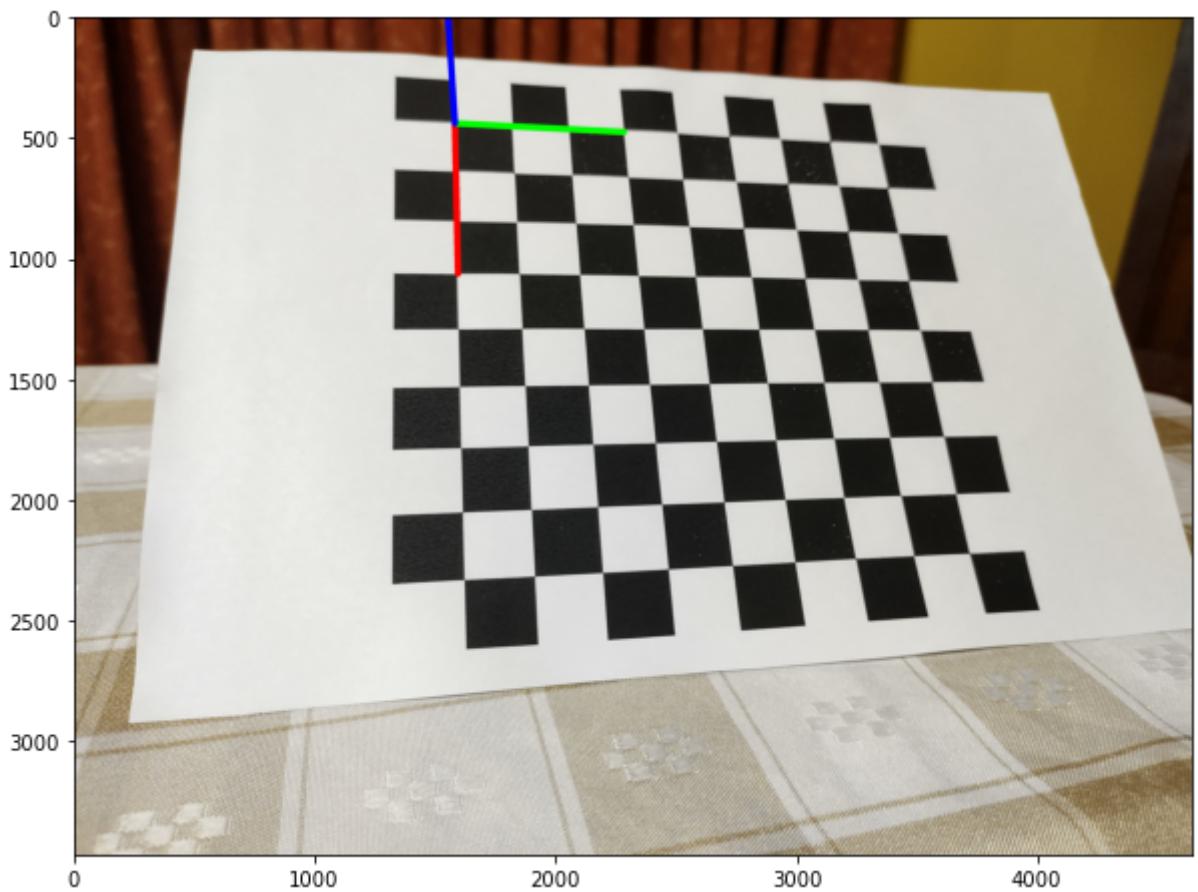
Dibujando en la imagen 3



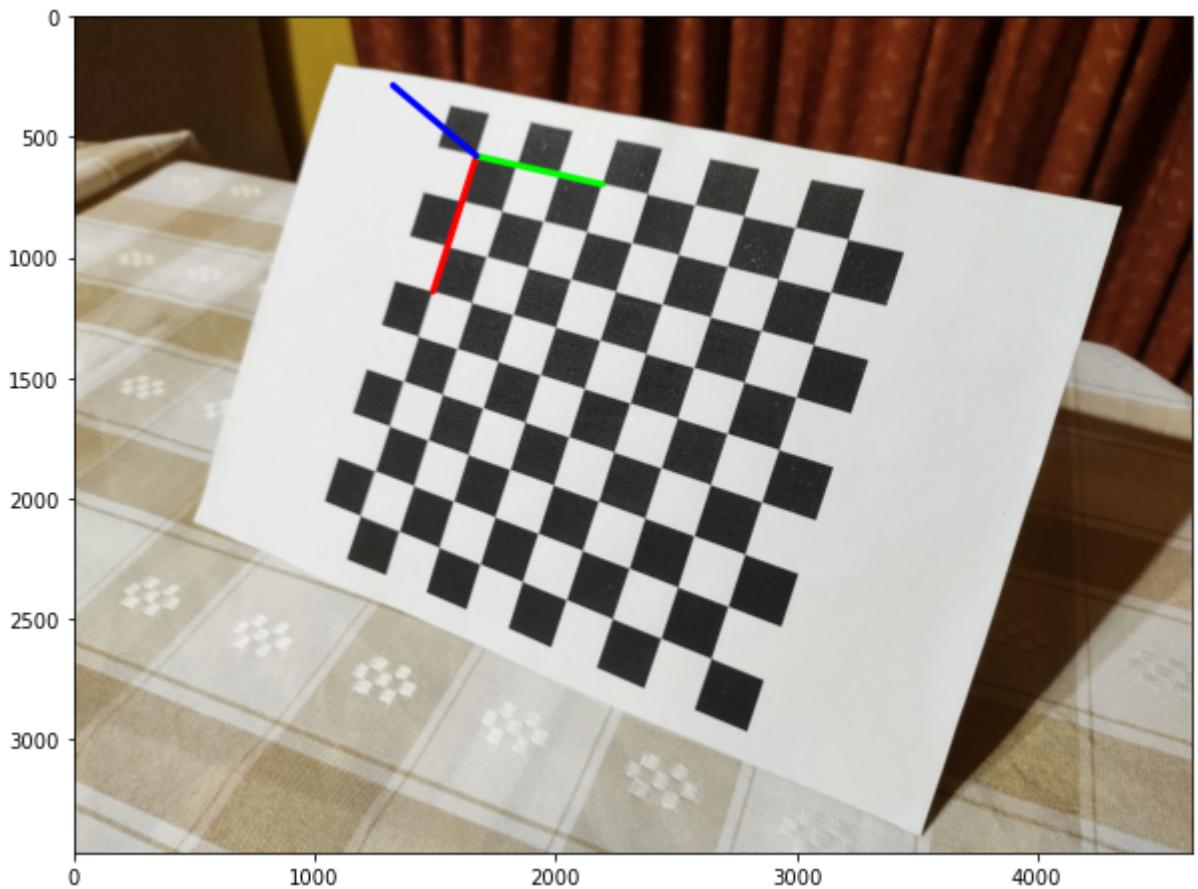
Dibujando en la imagen 4



Dibujando en la imagen 5



Dibujando en la imagen 6



Dibujando lineas de un cubo

In [21]:

```
def dibujar_cubo(idx, img, esq_3d):  
    esquinas_2d, _ = cv2.projectPoints(esq_3d, rvecs[idx], tvecs[idx], mtx, dist)
```

```

esq_2d = esquinas_2d[:,0, :]

r = (0, 0, 255) # r (in BGR)
b = (255, 0, 0) # b (in BGR)
g = (0, 255, 0) # g (in BGR)
line_width = 15

# Redondeamos los valores de los puntos porque cv.Line no toma puntos en np.float
esq_round = []
for esquina in esq_2d:
    x = round(esquina[0])
    y = round(esquina[1])

    round_points = (x,y)
    esq_round.append(round_points)

# base
cv2.line(img, esq_round[0], esq_round[1], r, line_width)
cv2.line(img, esq_round[1], esq_round[2], r, line_width)
cv2.line(img, esq_round[2], esq_round[3], r, line_width)
cv2.line(img, esq_round[3], esq_round[0], r, line_width)

# pilares
cv2.line(img, esq_round[0], esq_round[4], b, line_width)
cv2.line(img, esq_round[1], esq_round[5], b, line_width)
cv2.line(img, esq_round[2], esq_round[6], b, line_width)
cv2.line(img, esq_round[3], esq_round[7], b, line_width)

# tapa
cv2.line(img, esq_round[4], esq_round[5], g, line_width)
cv2.line(img, esq_round[5], esq_round[6], g, line_width)
cv2.line(img, esq_round[6], esq_round[7], g, line_width)
cv2.line(img, esq_round[7], esq_round[4], g, line_width)

plt.figure(figsize=(10,8))
plt.imshow(img[...,:-1])
plt.show()

```

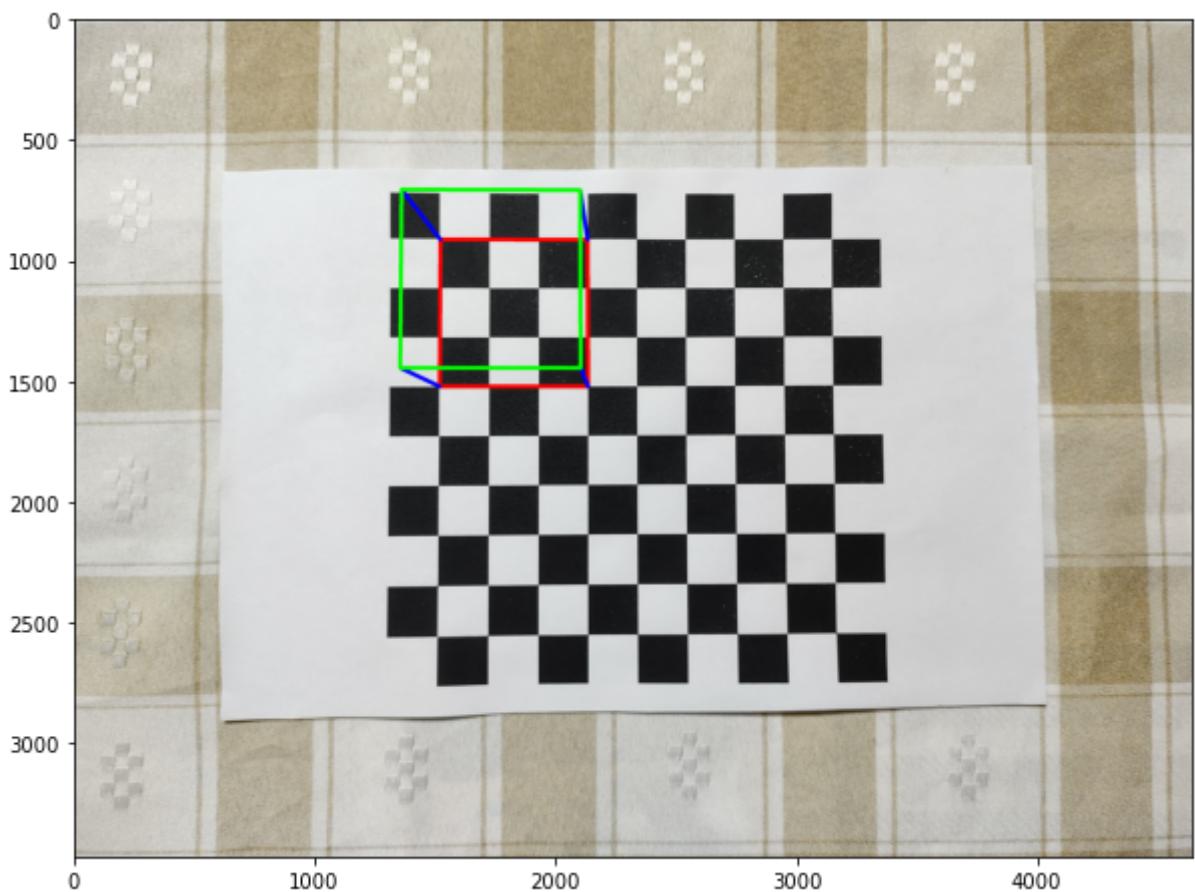
In [22]:

```

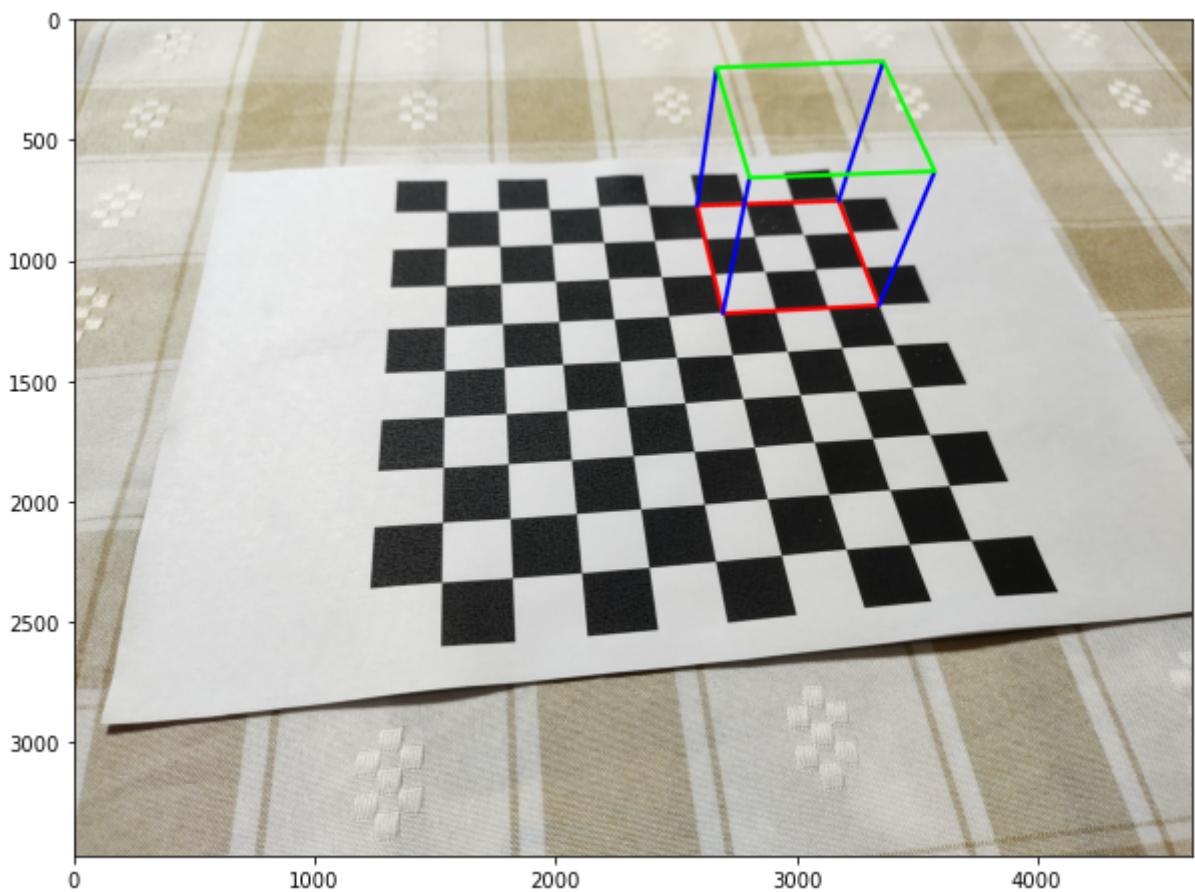
for i in range(6):
    img=cv2.imread(calib_fnames[i])
    print("Dibujando en la imagen ", i+1)
    dibujar_cubo(i, img, esquinas1)

```

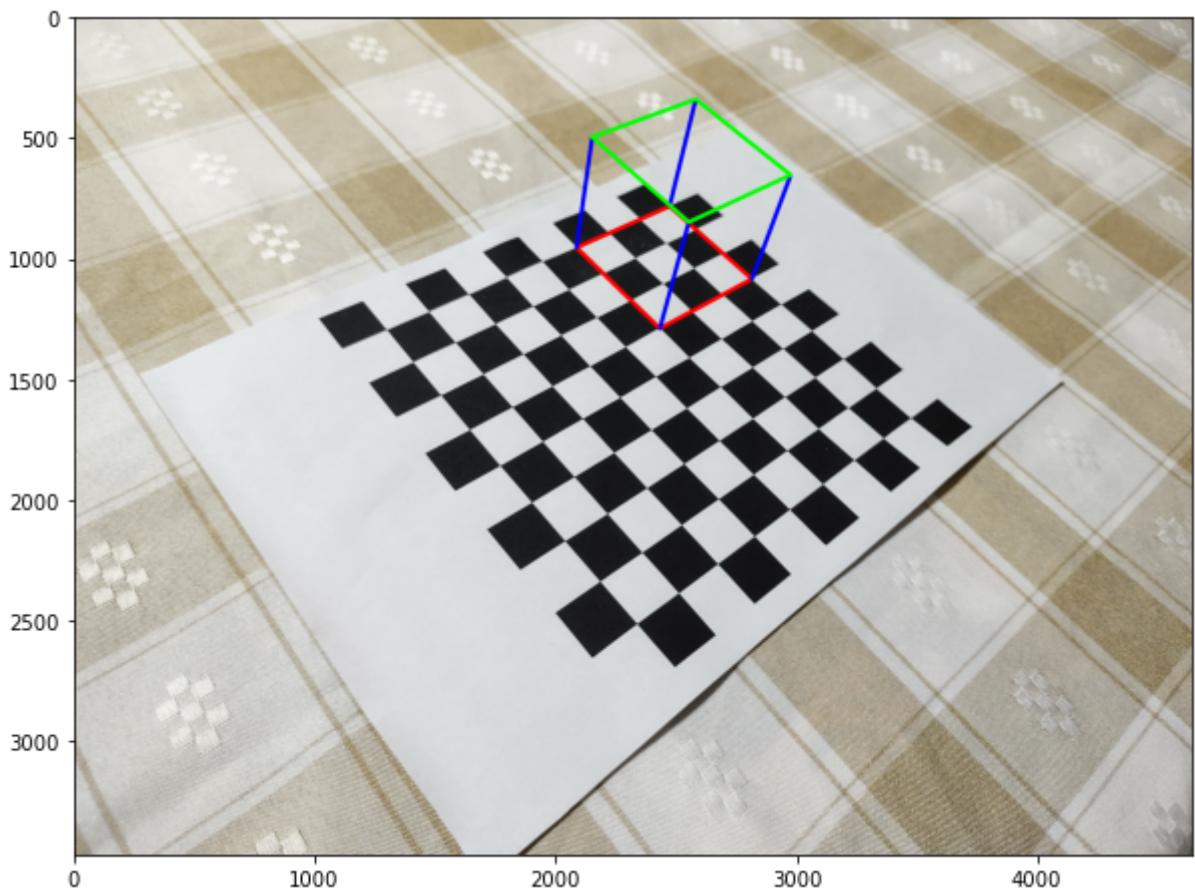
Dibujando en la imagen 1



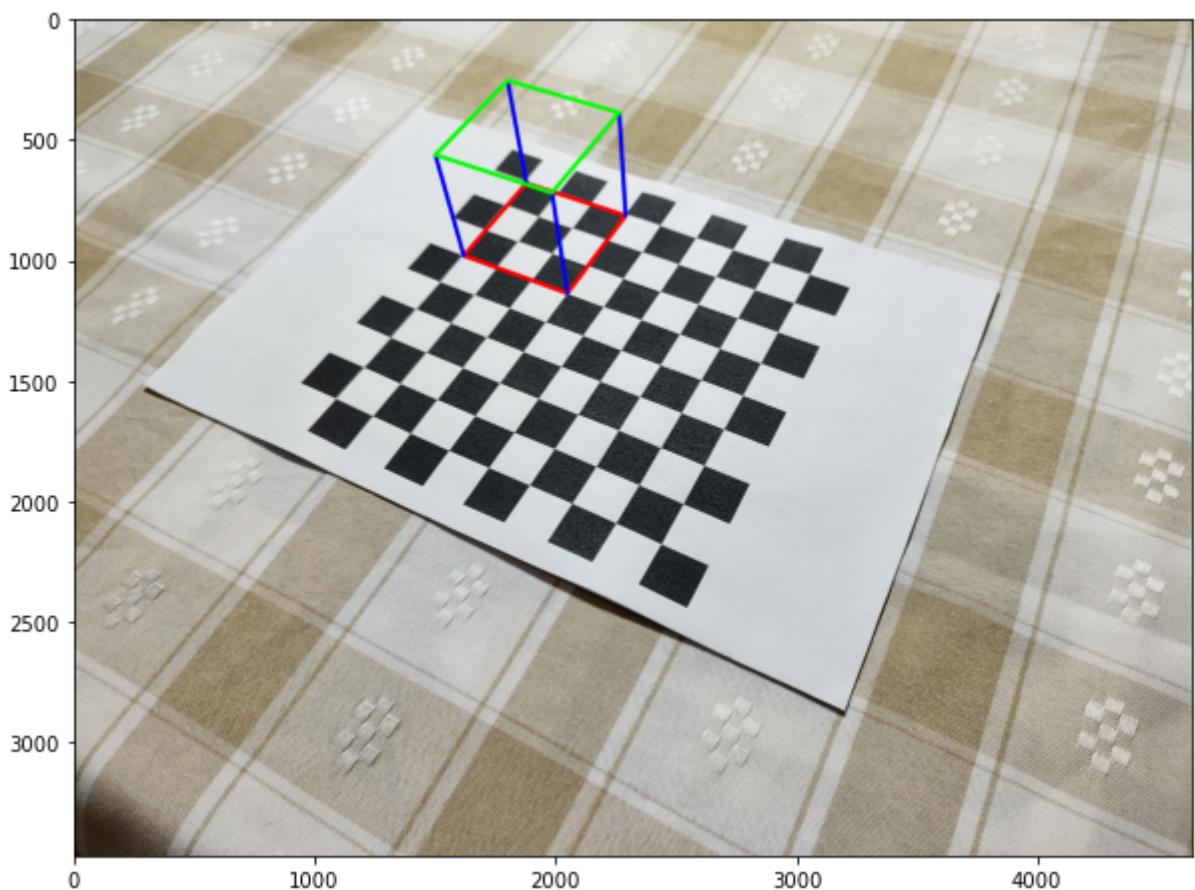
Dibujando en la imagen 2



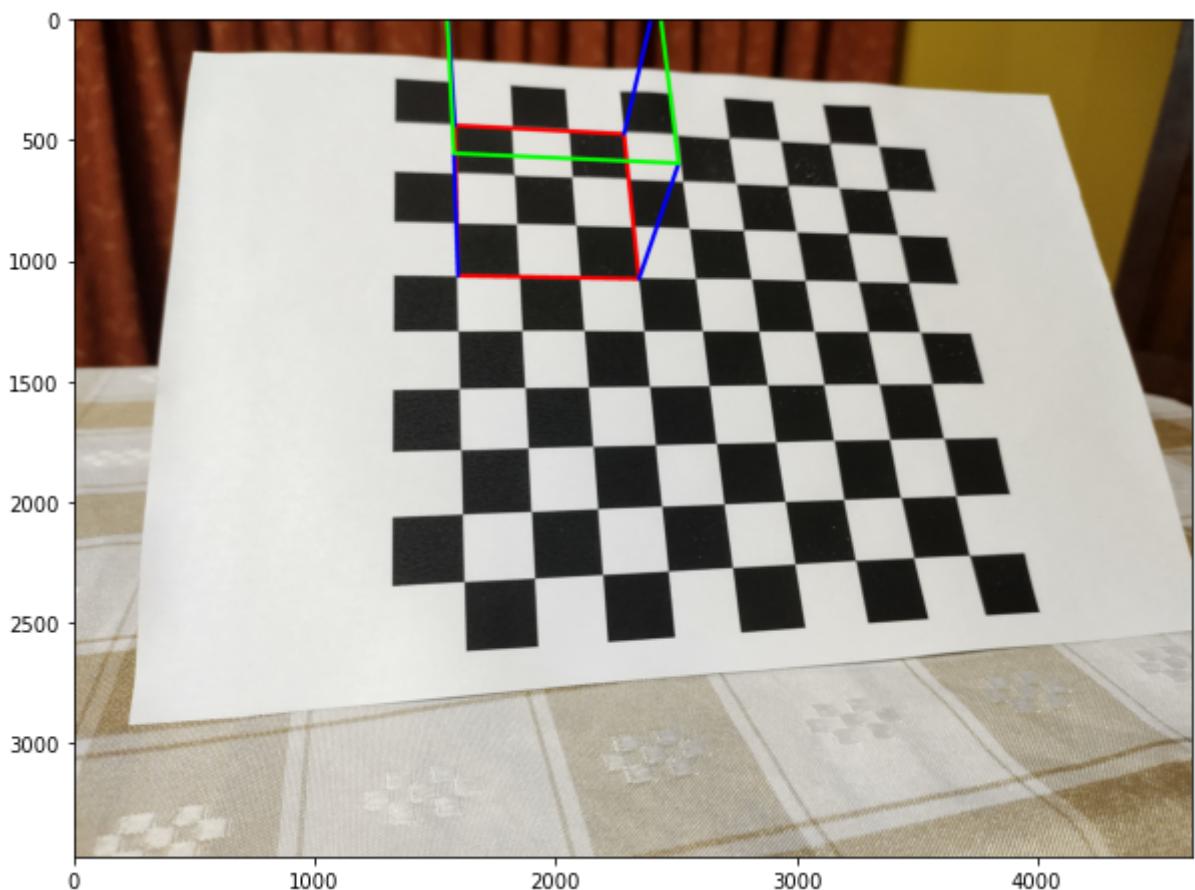
Dibujando en la imagen 3



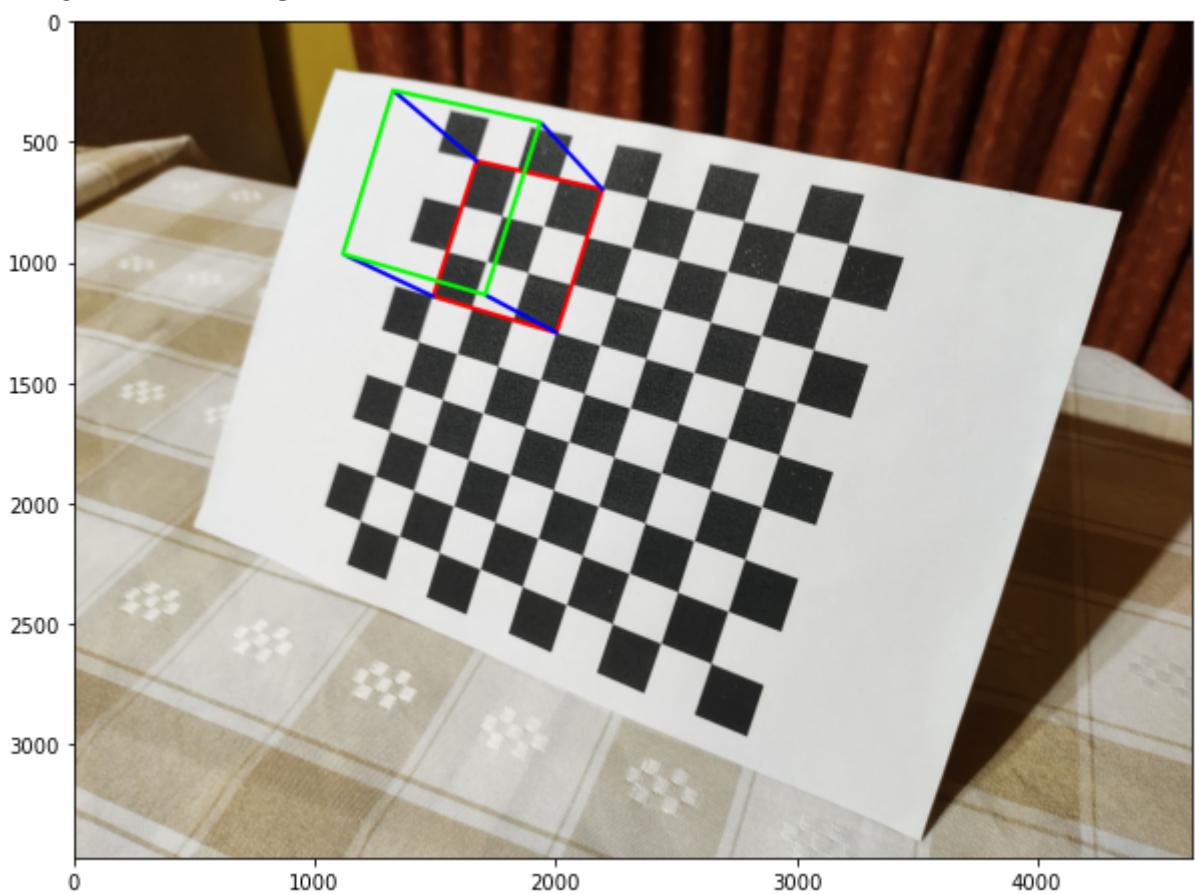
Dibujando en la imagen 4



Dibujando en la imagen 5



Dibujando en la imagen 6



Dibujando cubo con base de color liso

```
In [23]: # Dibujo en la primera imagen  
idx = 0
```

```

esquinas_2d, _ = cv2.projectPoints(esquinas1,rvecs[idx],tvecs[idx],mtx,dist)
esq_2d = esquinas_2d[:,0, :]

r = (0, 0, 255) # r (in BGR)
b = (255, 0, 0) # b (in BGR)
g = (0, 255, 0) # g (in BGR)
line_width = 15

img=cv2.imread(calib_fnames[idx])

#Redondeamos los valores de los puntos porque cv.line no toma puntos en np.float32
esq_round = []
for esquina in esq_2d:
    x = round(esquina[0])
    y = round(esquina[1])

    round_points = (x,y)
    esq_round.append(round_points)

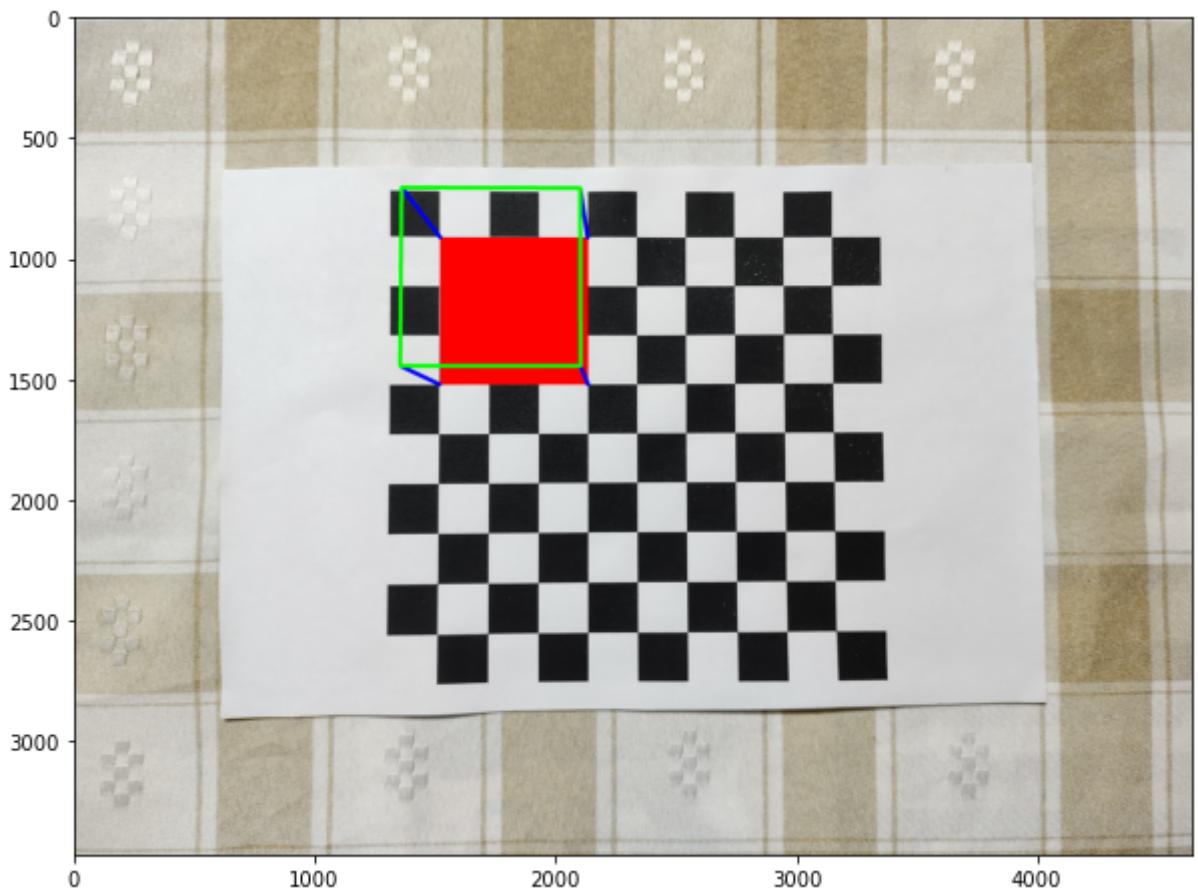
# base
cv2.rectangle(img, esq_round[0], esq_round[2], r, -1)

# pilares
cv2.line(img, esq_round[0], esq_round[4], b, line_width)
cv2.line(img, esq_round[1], esq_round[5], b, line_width)
cv2.line(img, esq_round[2], esq_round[6], b, line_width)
cv2.line(img, esq_round[3], esq_round[7], b, line_width)

# tapa
cv2.line(img, esq_round[4], esq_round[5], g, line_width)
cv2.line(img, esq_round[5], esq_round[6], g, line_width)
cv2.line(img, esq_round[6], esq_round[7], g, line_width)
cv2.line(img, esq_round[7], esq_round[4], g, line_width)

plt.figure(figsize=(10,8))
plt.imshow(img[...,:-1])
plt.show()

```

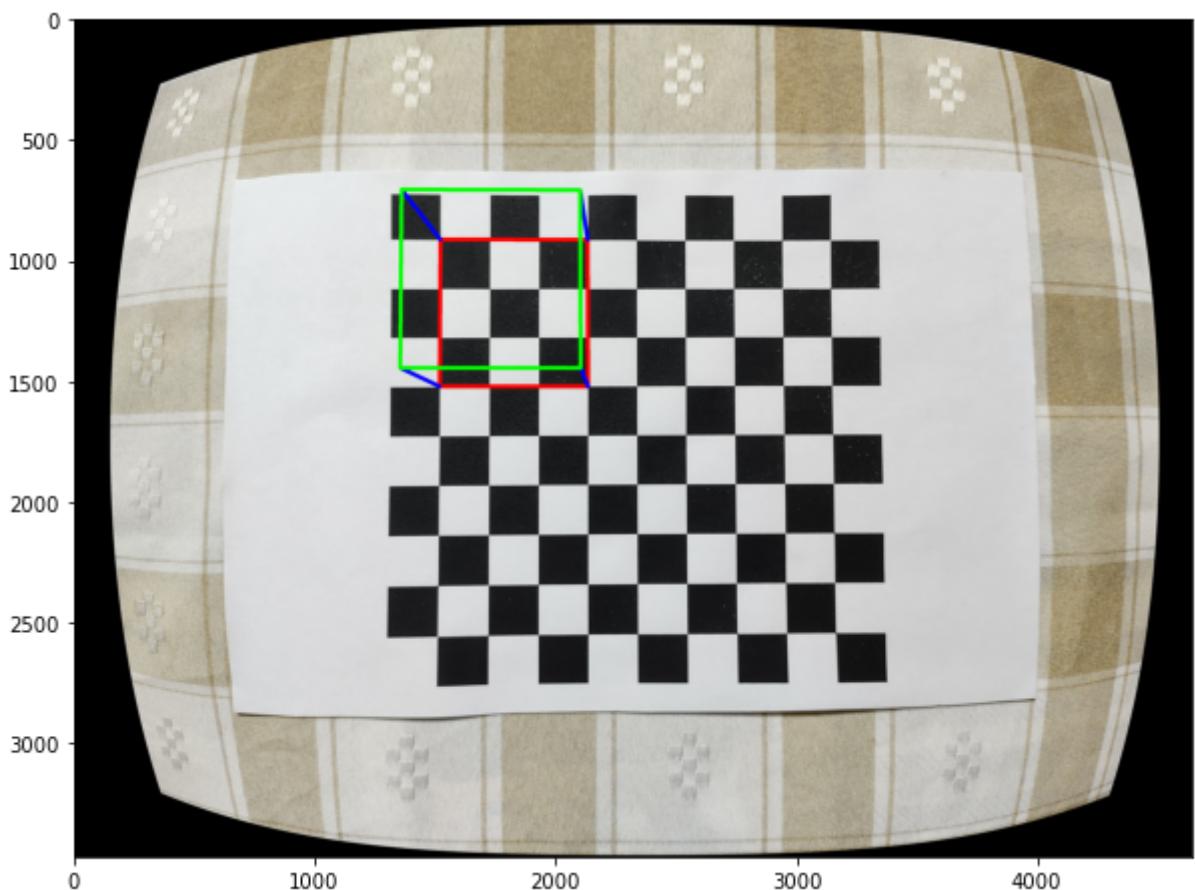


Dibujamos un cubo en las imágenes rectificadas

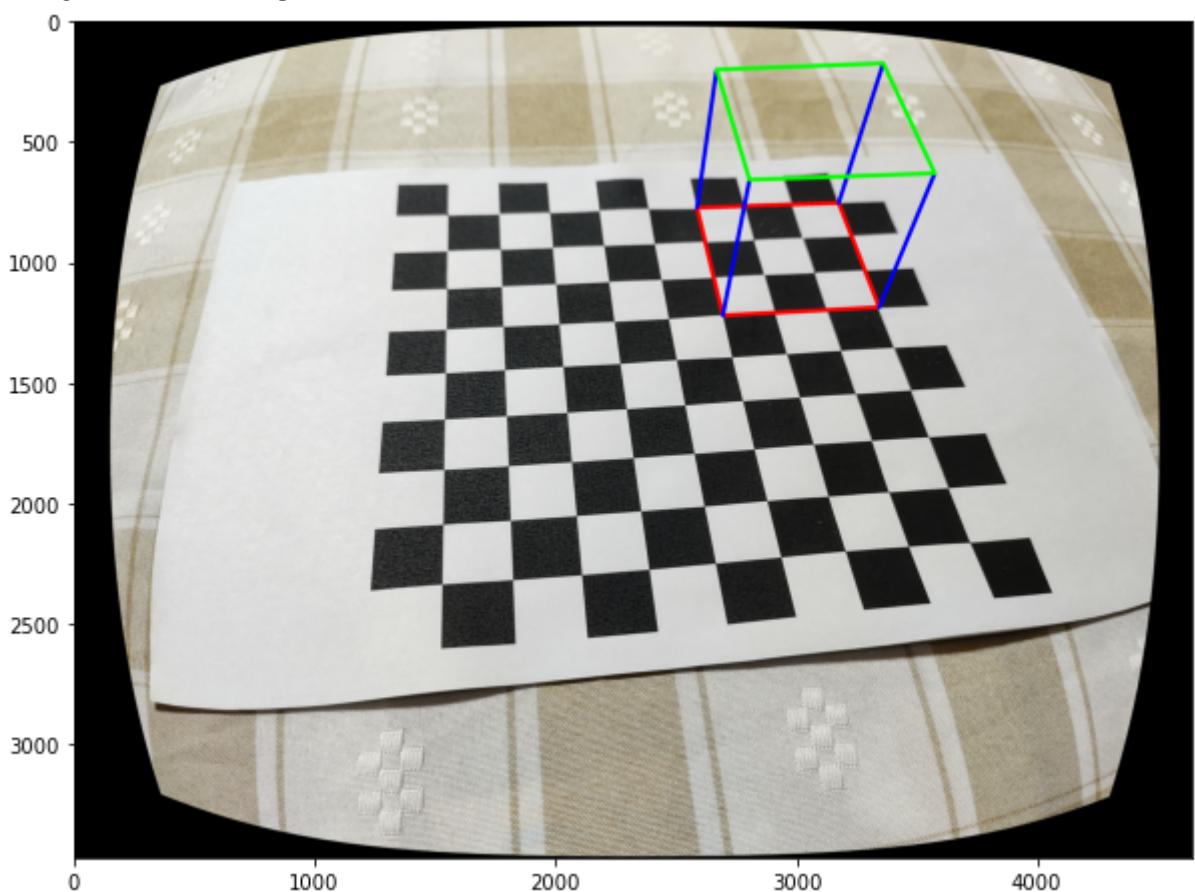
In [24]:

```
for i in range(6):
    img=imgs_dst[i]
    print("Dibujando en la imagen ", i+1)
    dibujar_cubo(i, img, esquinas1)
```

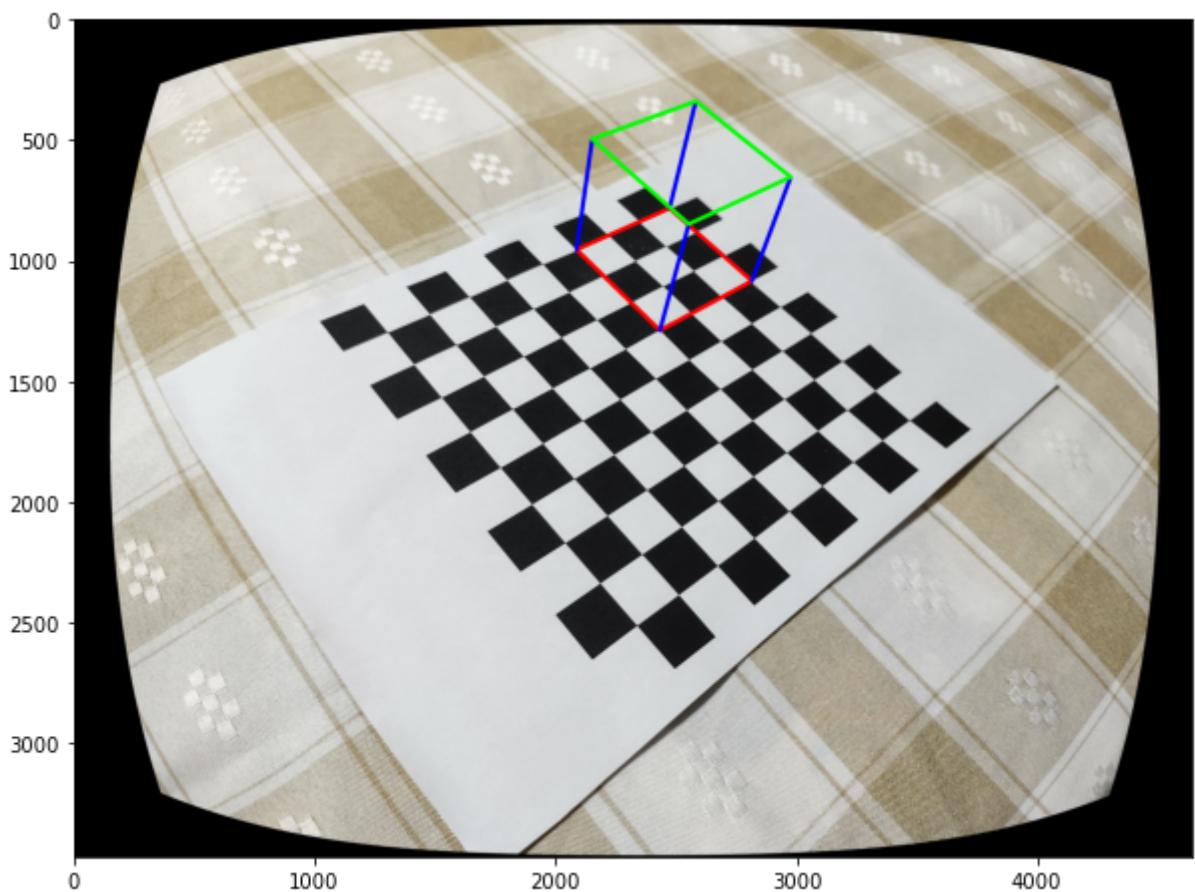
Dibujando en la imagen 1



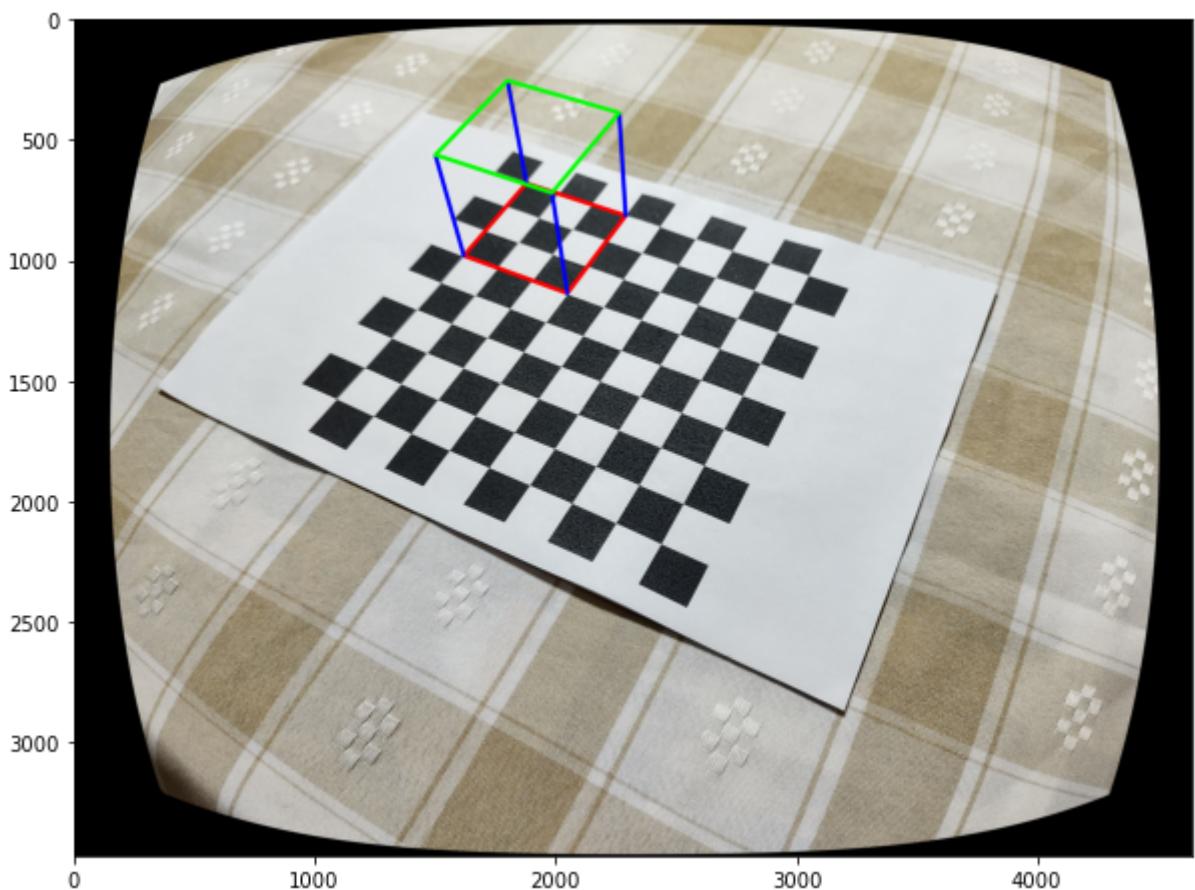
Dibujando en la imagen 2



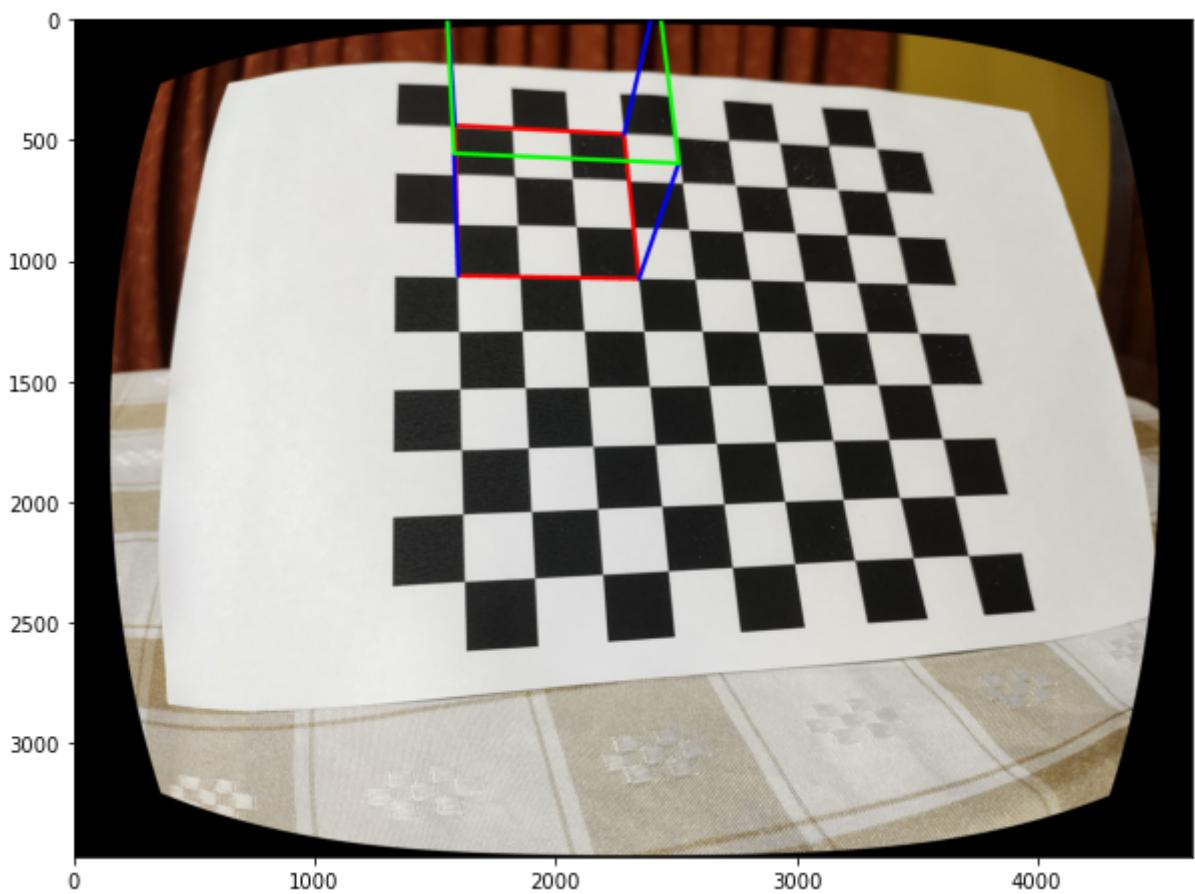
Dibujando en la imagen 3



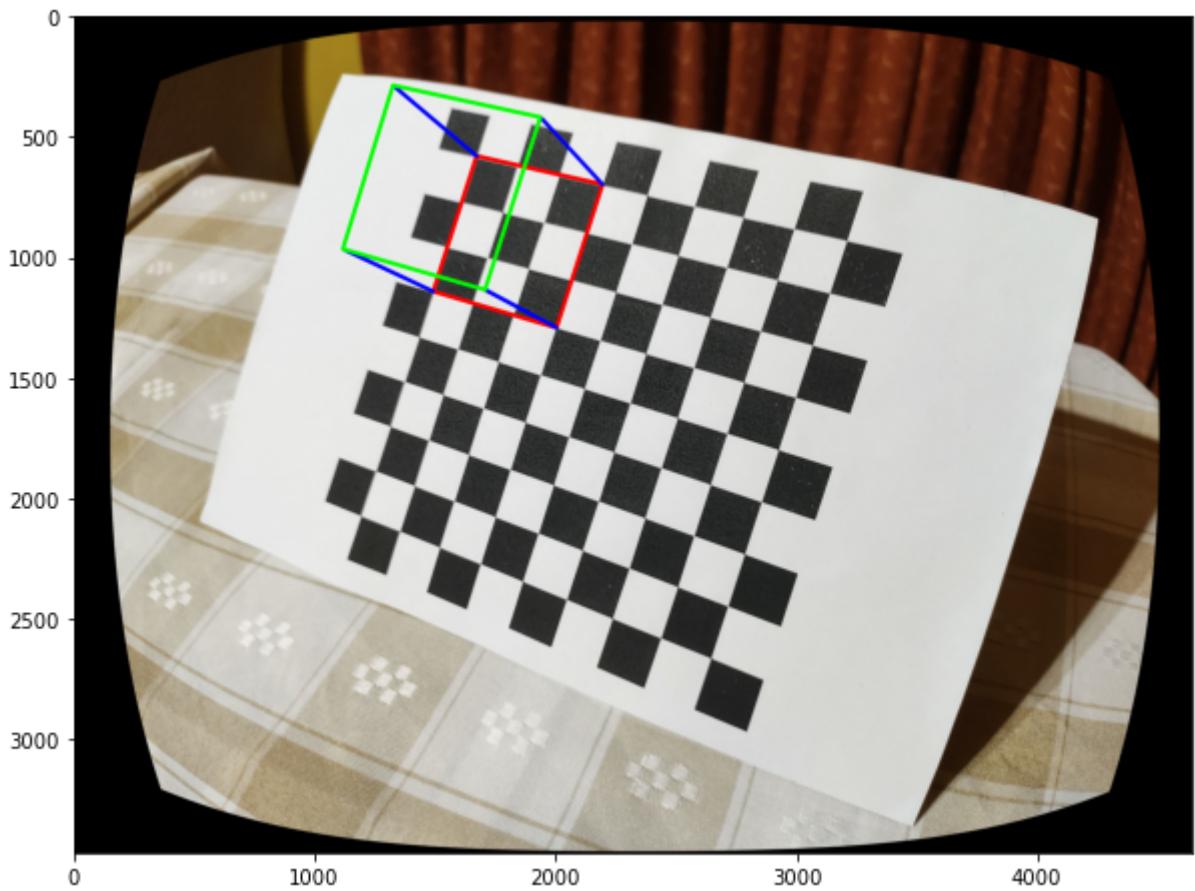
Dibujando en la imagen 4



Dibujando en la imagen 5



Dibujando en la imagen 6



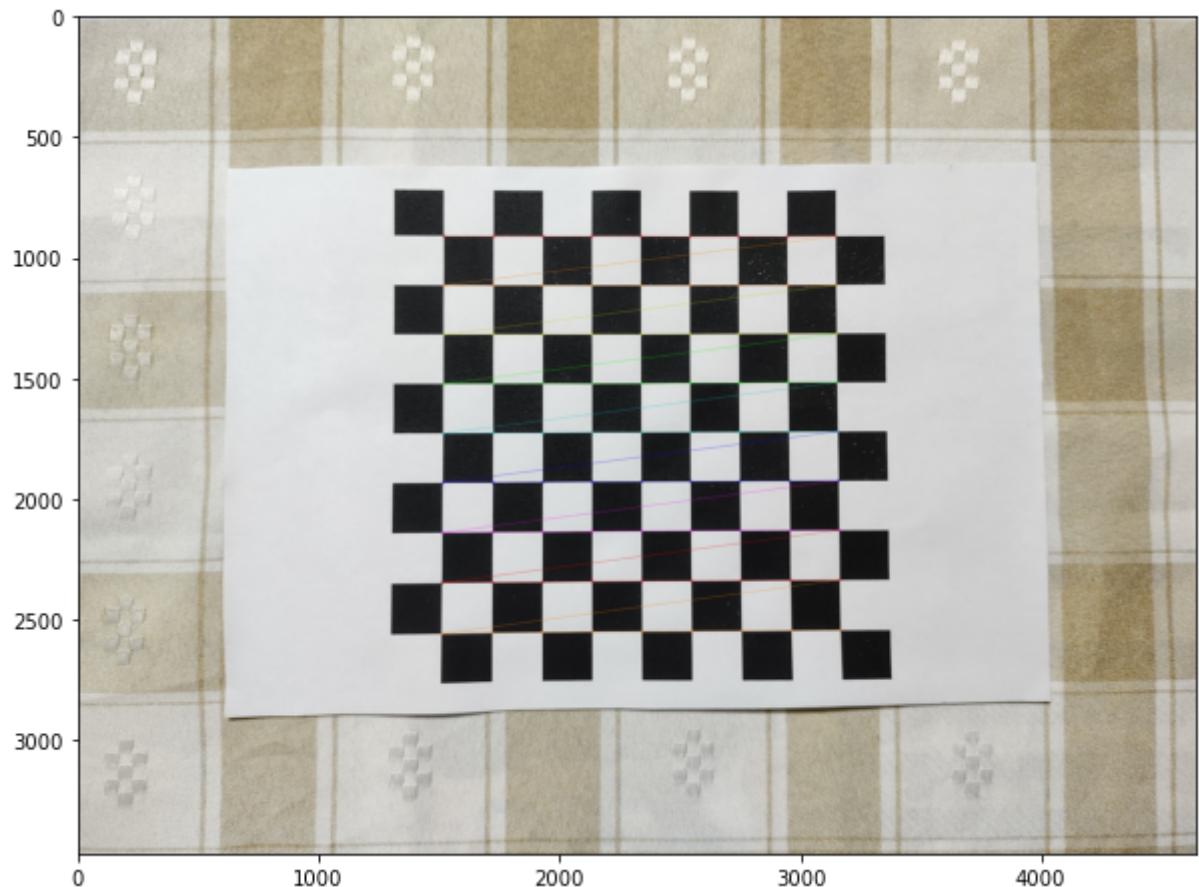
Cambiando iteraciones en criteria de cornerSubPix

Con menos iteraciones

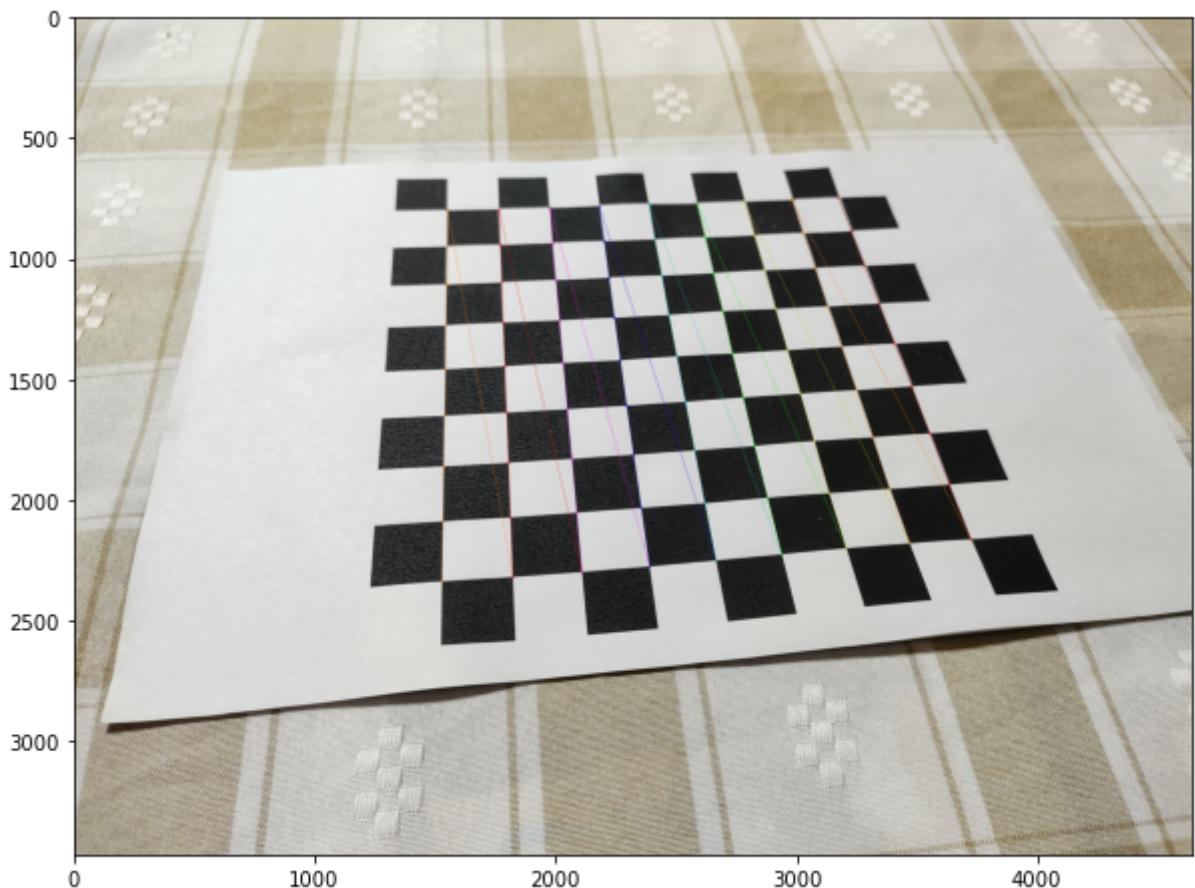
```
In [25]: obj_points, img_points = list(), list()
maxCount = 6

encontrar_esquinas(cv2.CALIB_CB_FILTER_QUADS)
```

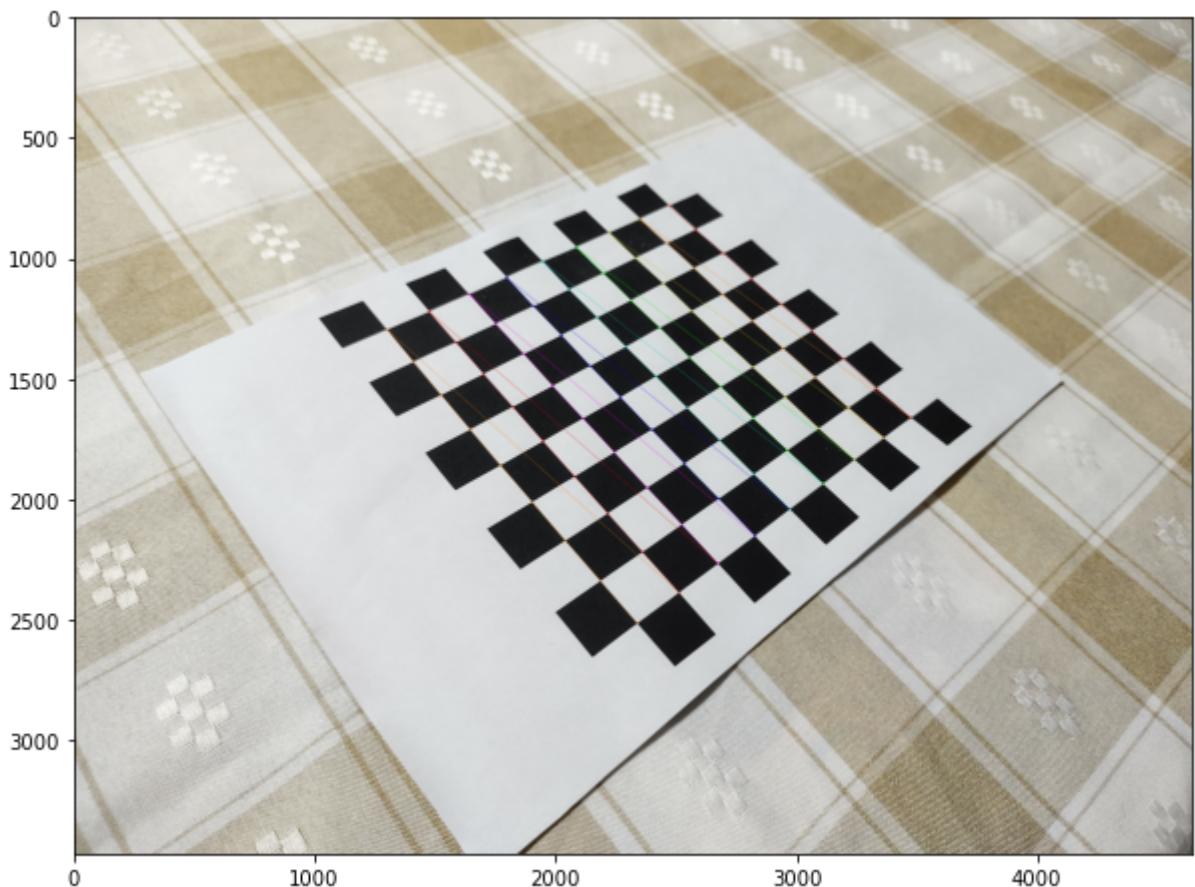
Procesando: ./fotos\Imagen 01.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



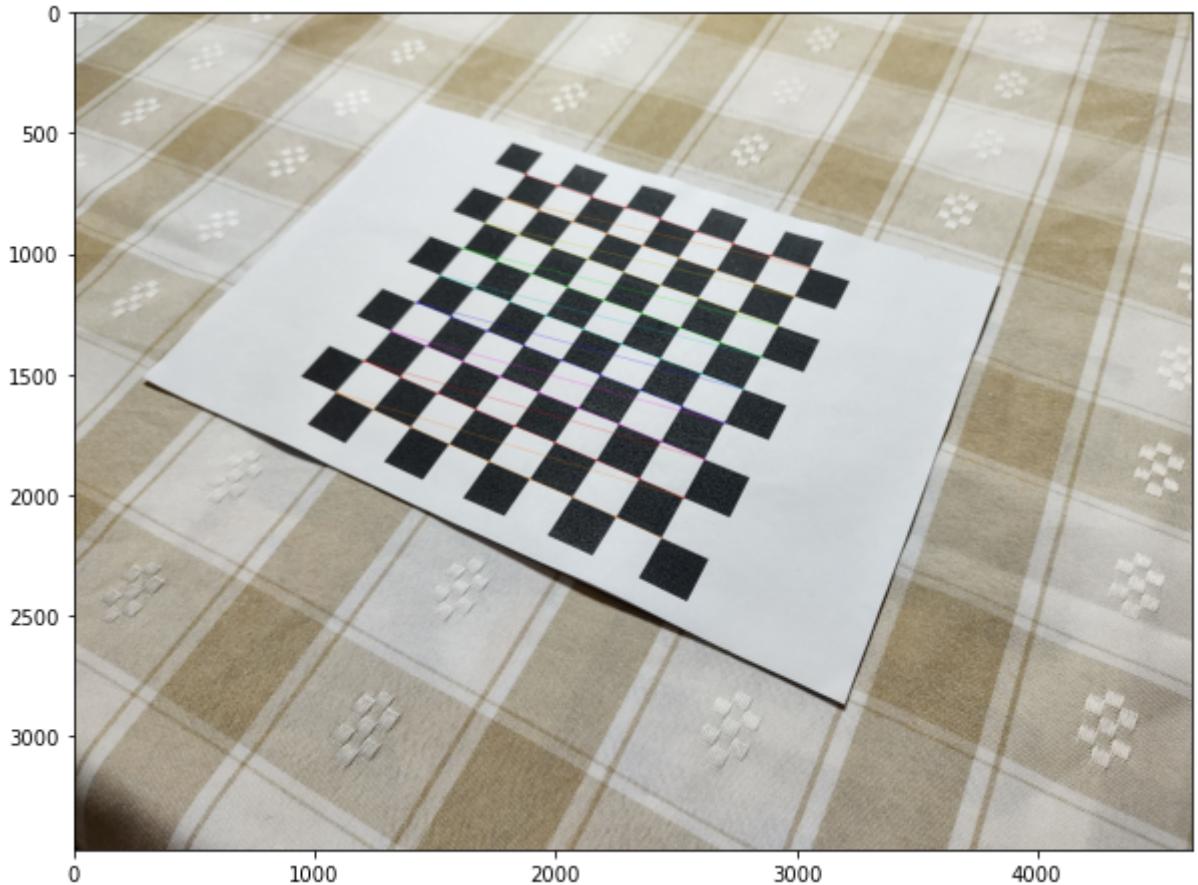
Procesando: ./fotos\Imagen 02.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



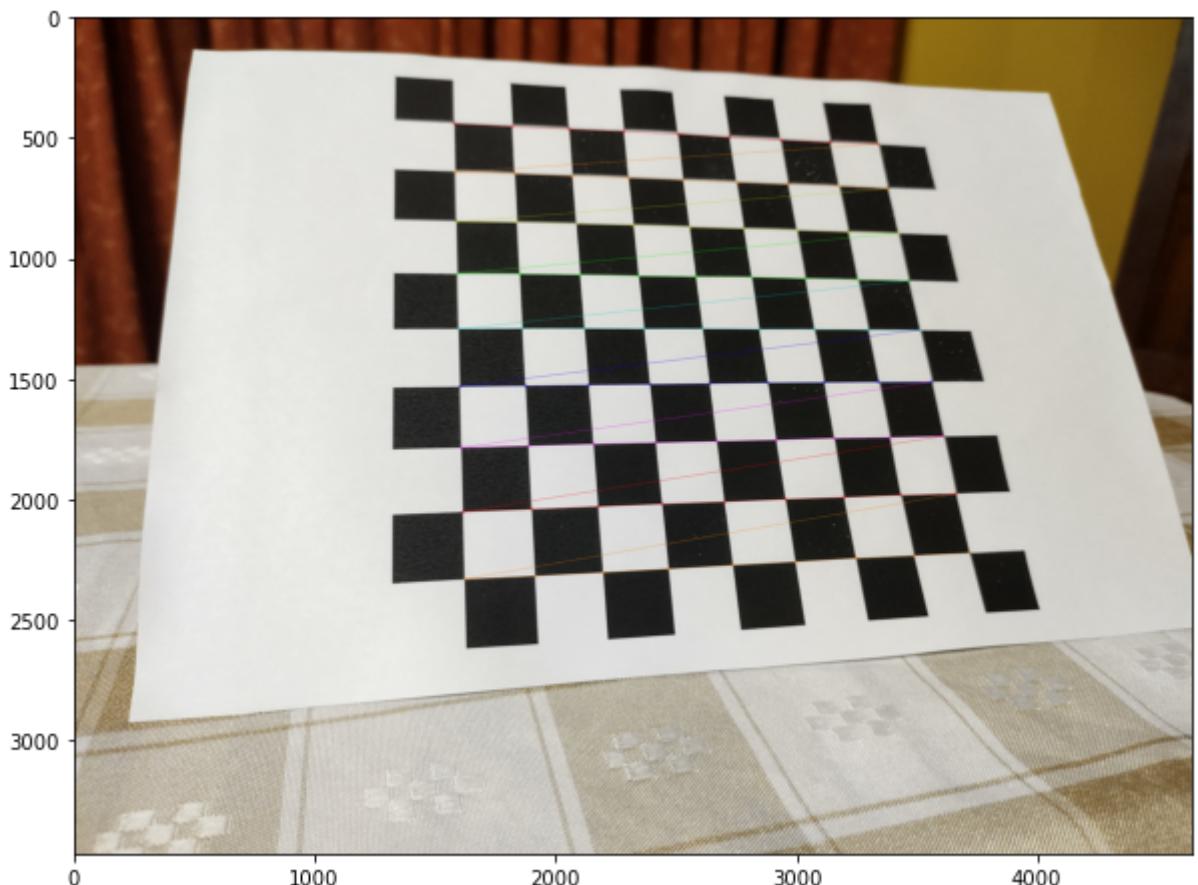
Procesando: ./fotos\Imagen 03.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



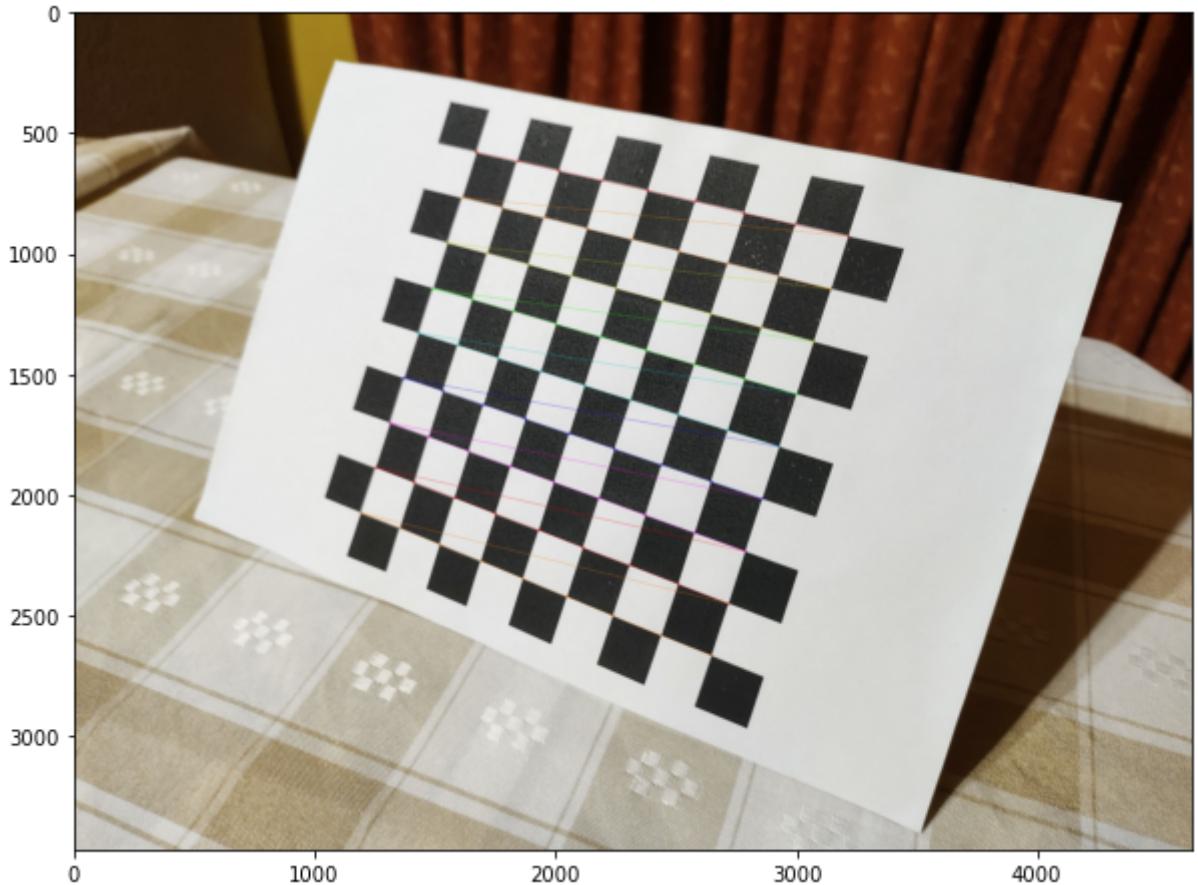
Procesando: ./fotos\Imagen 04.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 05.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 06.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



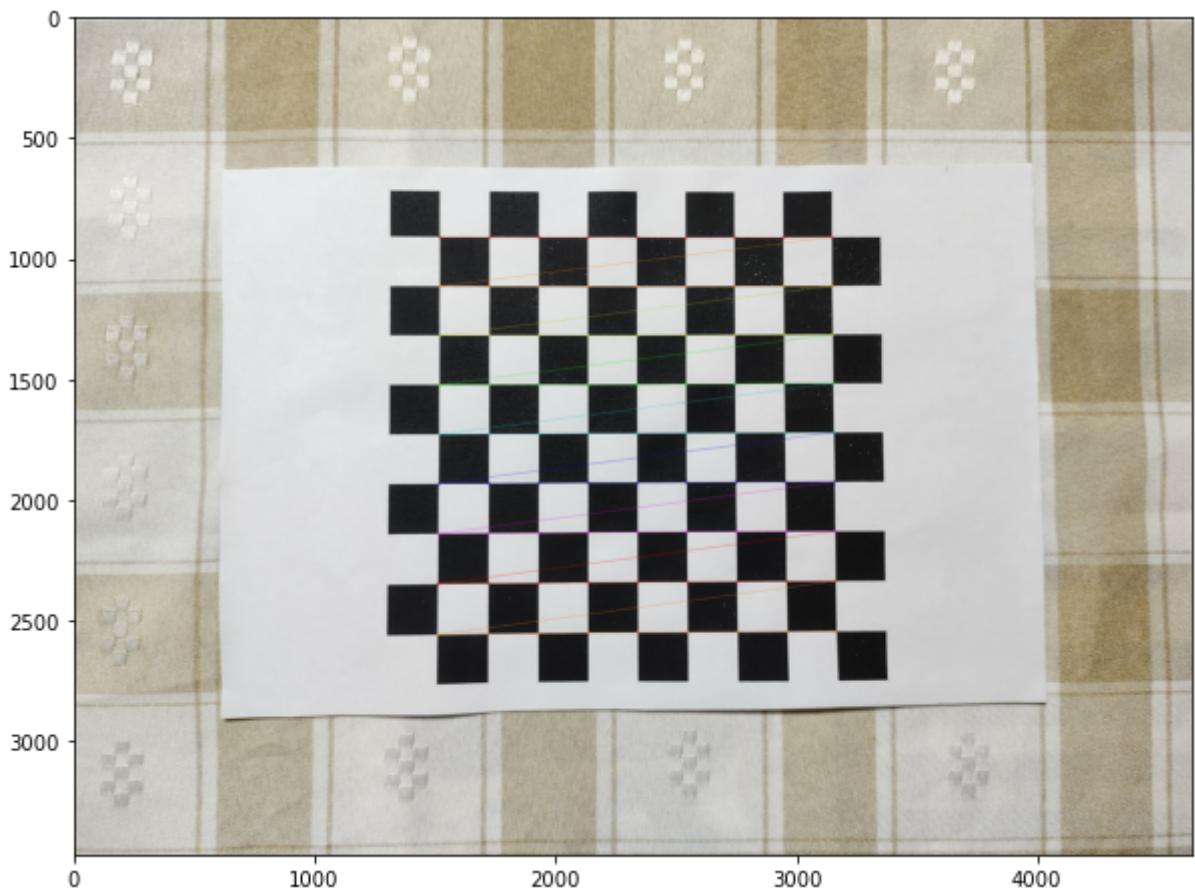
Procesando: ./fotos\Imagen 07.jpg... No se encontraron esquinas
Procesando: ./fotos\Imagen 08.jpg... No se encontraron esquinas

Tiempo de procesamiento 47.46348690986633 segundos

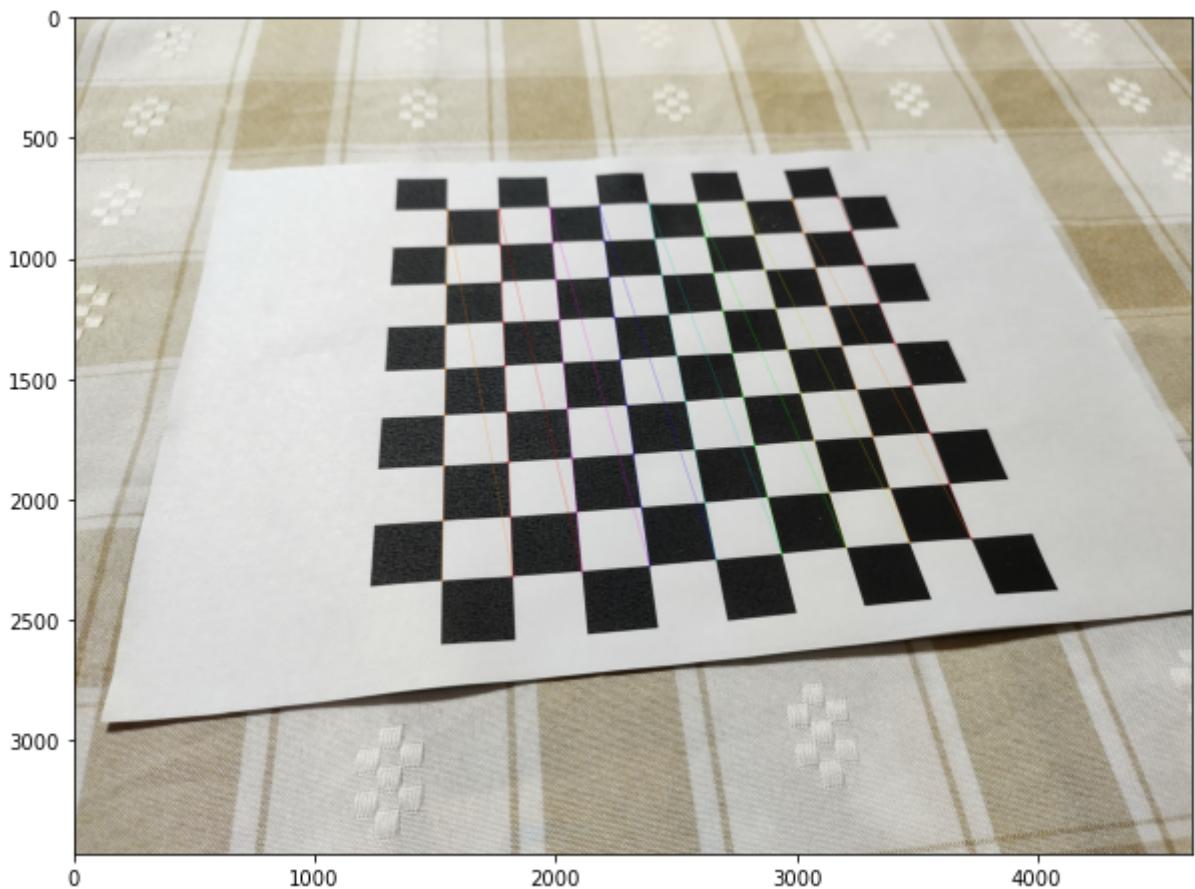
Con mas iteraciones

```
In [26]:  
obj_points, img_points = list(), list()  
maxCount = 40  
  
encontrar_esquinas(cv2.CALIB_CB_FILTER_QUADS)
```

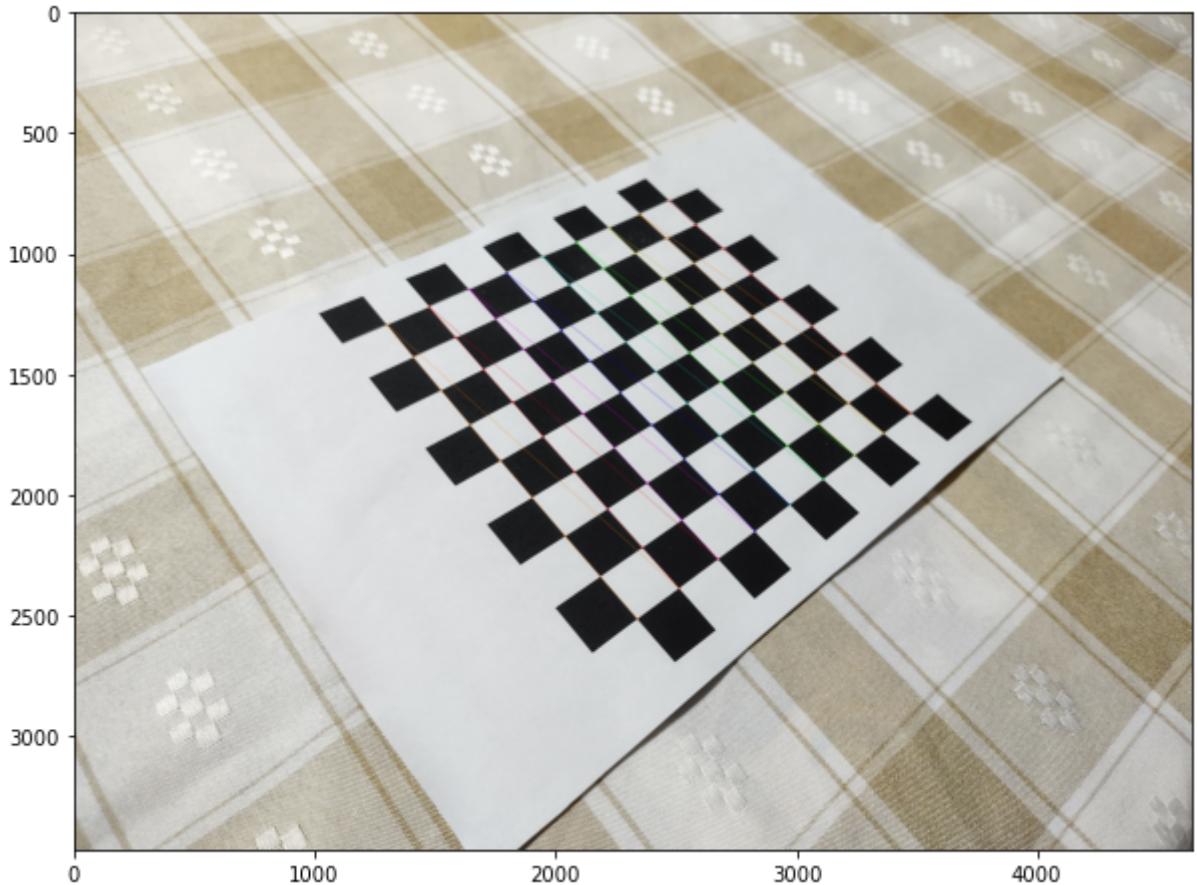
Procesando: ./fotos\Imagen 01.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



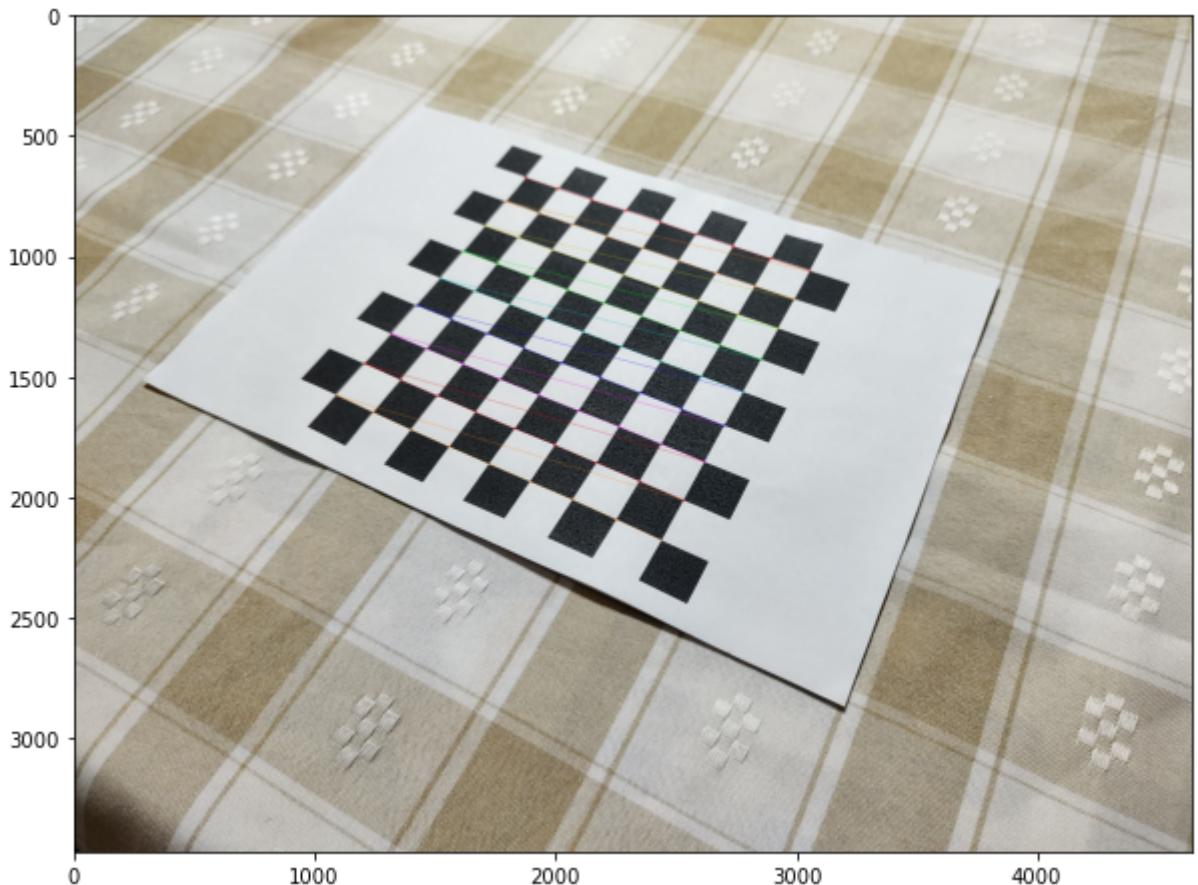
Procesando: ./fotos\Imagen 02.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



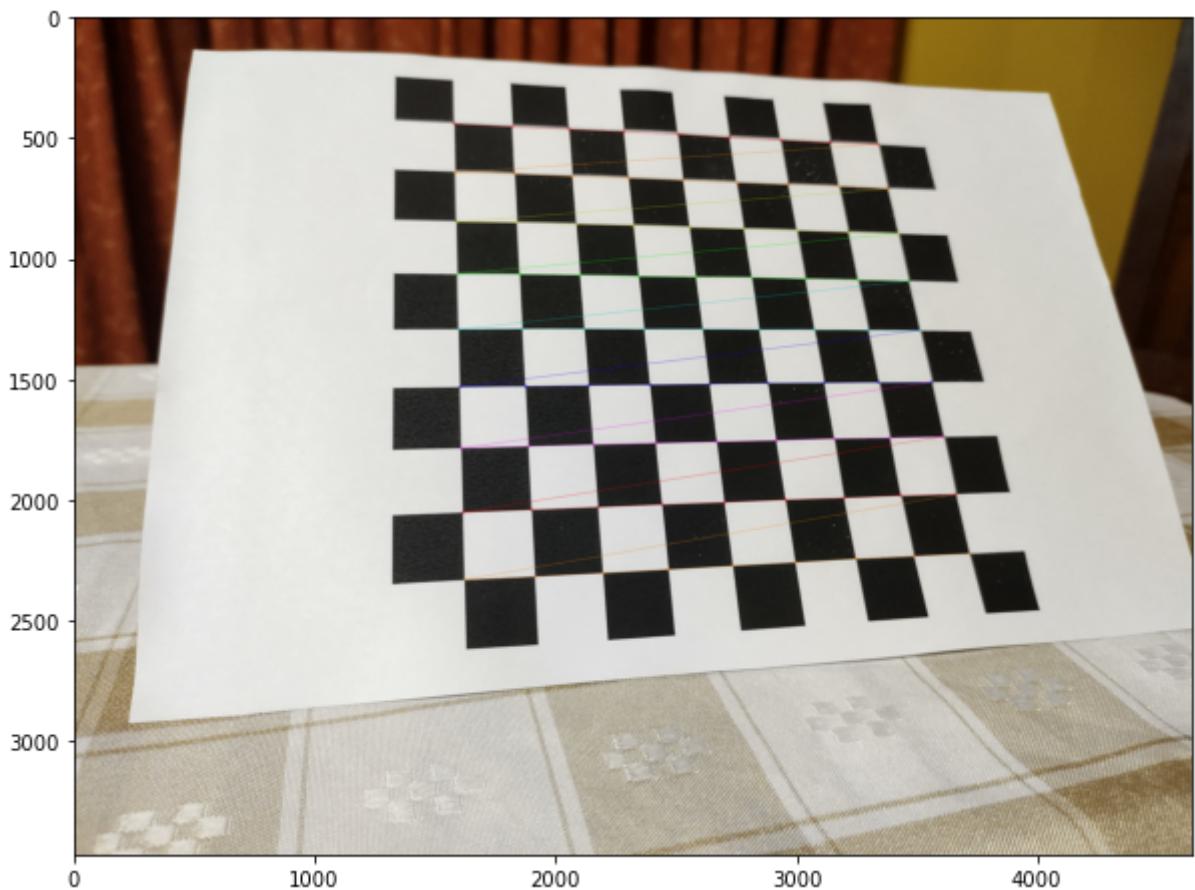
Procesando: ./fotos\Imagen 03.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



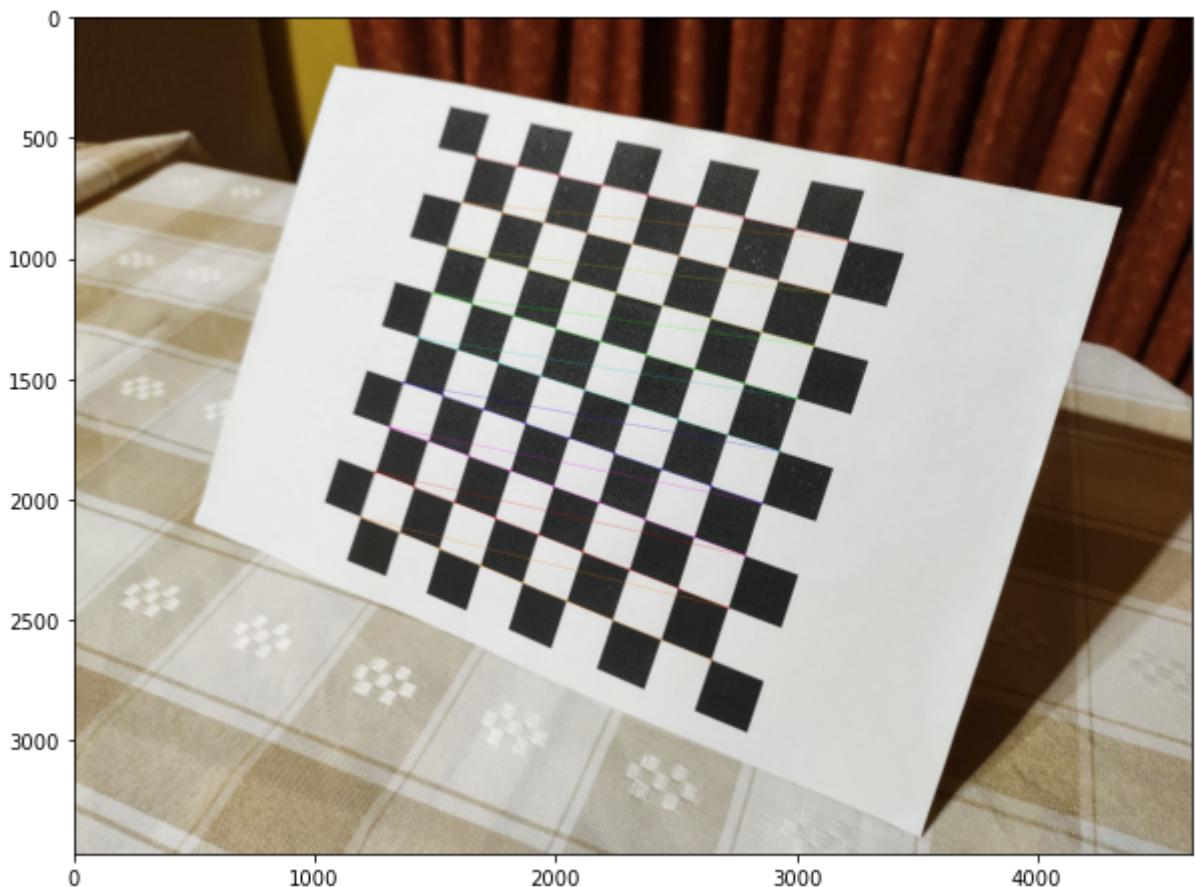
Procesando: ./fotos\Imagen 04.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 05.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 06.jpg... Encontramos esquinas!
Buscando esquinas en resolución subpixel... OK!



Procesando: ./fotos\Imagen 07.jpg... No se encontraron esquinas
Procesando: ./fotos\Imagen 08.jpg... No se encontraron esquinas

Tiempo de procesamiento 60.6174521446228 segundos

Tanto con mas o menos iteraciones obtenemos buenos resultados, la diferencia se observa en el

tiempo de procesamiento.

Usando otros flags en calibrateCamera

In [27]:

```
flags = ['cv2.CALIB_USE_INTRINSIC_GUESS', 'cv2.CALIB_FIX_PRINCIPAL_POINT', 'cv2.CALIB_ZERO_TANGENT_DIST', 'cv2.CALIB_RATIONAL_MODEL', 'cv2.CALIB_TI  
for f in flags:  
  
    print(f)  
    flag = eval(f)  
  
    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points, (w, h),  
                                                    flag)  
  
    print('\nMatriz de la camara = ')  
    print(mtx)  
  
    print('\nCoeficientes de distorsion = ')  
    print(dist)  
  
    print('-----')
```

cv2.CALIB_USE_INTRINSIC_GUESS

Matriz de la camara =
[[3.46785470e+03 0.00000000e+00 2.34241813e+03]
[0.00000000e+00 3.46700670e+03 1.72400515e+03]
[0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Coeficientes de distorsion =
[[1.17215277e-01]
[-1.11188740e+00]
[-3.44442680e-03]
[7.36021141e-04]
[3.13644405e+00]]

cv2.CALIB_FIX_PRINCIPAL_POINT

Matriz de la camara =
[[3.46785470e+03 0.00000000e+00 2.34241813e+03]
[0.00000000e+00 3.46700670e+03 1.72400515e+03]
[0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Coeficientes de distorsion =
[[1.17215277e-01]
[-1.11188740e+00]
[-3.44442680e-03]
[7.36021141e-04]
[3.13644405e+00]]

cv2.CALIB_FIX_ASPECT_RATIO

Matriz de la camara =
[[3.46785470e+03 0.00000000e+00 2.34241813e+03]
[0.00000000e+00 3.46700670e+03 1.72400515e+03]
[0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Coeficientes de distorsion =
[[1.17215277e-01]
[-1.11188740e+00]
[-3.44442680e-03]
[7.36021141e-04]
[3.13644405e+00]]

cv2.CALIB_ZERO_TANGENT_DIST

```
Matriz de la camara =
[[3.46785470e+03 0.00000000e+00 2.34241813e+03]
 [0.00000000e+00 3.46700670e+03 1.72400515e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

```
Coeficientes de distorsion =
[[ 1.17215277e-01]
 [-1.11188740e+00]
 [-3.44442680e-03]
 [ 7.36021141e-04]
 [ 3.13644405e+00]]
```

```
cv2.CALIB_RATIONAL_MODEL
```

```
Matriz de la camara =
[[3.46785470e+03 0.00000000e+00 2.34241813e+03]
 [0.00000000e+00 3.46700670e+03 1.72400515e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

```
Coeficientes de distorsion =
[[ 1.17215277e-01]
 [-1.11188740e+00]
 [-3.44442680e-03]
 [ 7.36021141e-04]
 [ 3.13644405e+00]]
```

```
cv2.CALIB_TILTED_MODEL
```

```
Matriz de la camara =
[[3.46785470e+03 0.00000000e+00 2.34241813e+03]
 [0.00000000e+00 3.46700670e+03 1.72400515e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

```
Coeficientes de distorsion =
[[ 1.17215277e-01]
 [-1.11188740e+00]
 [-3.44442680e-03]
 [ 7.36021141e-04]
 [ 3.13644405e+00]]
```

Cada flag tiene un metodo distinto de proceso, pero aun asi con todos ellos se obtiene el mismo resultado tanto en la matriz de la camara como en los coeficientes de distorsion.