In [1]:
```
import numpy as np
import cv2
import glob

checkerboard_dims = (8, 6)

corners = checkerboard_dims[0] * checkerboard_dims[1]
corners_x = checkerboard_dims[0]-1
corners_y = checkerboard_dims[1]-1
# the corners used in the context of objp creation refers to the number o

#Create objp as a zero array of shape (number of corners, 3), float32
objp = np.zeros((corners, 3), dtype=np.float32)
#Set the first two columns of objp to the coordinate grid of corners
for i in range(corners):
    objp[i, 0] = i % checkerboard_dims[1]
    objp[i, 1] = i // checkerboard_dims[1]

#Initialize objpoints as an empty list
objpoints = []
#Initialize imgpoints as an empty list
imgpoints = []

#Load all checkerboard images using glob ('path/to/images/*.jpg')
images_path = 'calibration_photos/*.jpg'
images_files = glob.glob(images_path)#glob is not working and it isn't pr
print(images_files)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001

for image_file in images_files:
    img = cv2.imread(image_file)

    #Read the image

    #Convert the image to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)



    # Find the chessboard corners in the grayscale image
    ret, corners = cv2.findChessboardCorners(gray, checkerboard_dims, Non

    # If found, add object points, image points (after refining them)
    if ret == True:

        #Append objp to objpoints
        objpoints.append(objp)

        #Refine corner positions using cornerSubPix
        refined_corners = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1


        #Append refined corners to imgpoints
        imgpoints.append(refined_corners)

        # Optionally, draw chessboard corners on the image
        # Optionally, display the image with drawn corners
        # Wait for a short period
        cv2.drawChessboardCorners(img, checkerboard_dims, refined_corners

        cv2.imshow('img', img)
        cv2.waitKey(500)


#Destroy all OpenCV windows
cv2.destroyAllWindows()

# Calibrate the camera using calibrateCamera with objpoints, imgpoints, a
# Get the camera matrix, distortion coefficients, rotation vectors, and t

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,

np.savez("calibration_matrix.npz", mtx, dist)

# Verify the calibration:
#     Initialize mean_error to 0
#     For each pair of object points and image points:
#         Project the object points to image points using projectPoints
#         Compute the error between the projected and actual image points
#         Accumulate the error
#     Compute the average error
#     Print the total average error
mean_error = 0
for i in range(len(objpoints)):
    imgpoints_proj, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i
    error = cv2.norm(imgpoints[i], imgpoints_proj, cv2.NORM_L2) / len(img
    mean_error += error

total_avg_error = mean_error / len(objpoints)
print("Total average error: ", total_avg_error)
```

```
['calibration_photos/calib19.jpg', 'calibration_photos/calib6.jpg', 'calib
ration_photos/calib10.jpg', 'calibration_photos/calib5.jpg', 'calibration_
photos/calib20.jpg', 'calibration_photos/calib12.jpg', 'calibration_photo
s/calib11.jpg', 'calibration_photos/calib9.jpg', 'calibration_photos/calib
16.jpg', 'calibration_photos/calib8.jpg', 'calibration_photos/calib14.jp
g', 'calibration_photos/calib15.jpg', 'calibration_photos/calib18.jpg', 'c
alibration_photos/calib4.jpg', 'calibration_photos/calib13.jpg', 'calibrat
ion_photos/calib17.jpg', 'calibration_photos/calib1.jpg', 'calibration_pho
tos/calib2.jpg', 'calibration_photos/calib3.jpg', 'calibration_photos/cali
b7.jpg']
```

```
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=w
ayland to run on Wayland anyway.
Total average error:  93.28172795275013
```

```
In [ ]:
```