In [1]:

```
import cv2
import numpy as np
import os
import glob
```

In [2]:

```
ref_image = cv2.imread('./cb_grids/images/grids_5by4/02.jpg')
ref_gray = cv2.cvtColor(ref_image,cv2.COLOR_BGR2GRAY)


CHECKERBOARD = (10,11)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

objpoints = []
imgpoints = []

objp = np.zeros((1, CHECKERBOARD[0] * CHECKERBOARD[1], 3), np.float32)
objp[0,:,:2] = np.mgrid[0:CHECKERBOARD[0], 0:CHECKERBOARD[1]].T.reshape(-1, 2)
prev_img_shape = None
```

In [3]:

```
# images = glob.glob('./cropped_images/*.png')

# for fname in images:
#     img = cv2.imread(fname)
#     img = cv2.resize(img, (0, 0), fx = 5.0, fy = 5.0)
#     gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#     # Color-segmentation to get binary mask
#     lwr = np.array([0, 0, 143])
#     upr = np.array([179, 61, 252])
#     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
#     msk = cv2.inRange(hsv, lwr, upr)

#     # Extract chess-board
#     krn = cv2.getStructuringElement(cv2.MORPH_RECT, (50, 30))
#     dlt = cv2.dilate(msk, krn, iterations=5)
#     res = 255 - cv2.bitwise_and(dlt, msk)

#     # Displaying chess-board features
#     res = np.uint8(res)

#     ret, corners = cv2.findChessboardCorners(res, CHECKERBOARD, flags = cv2.CALIB_CE
#     # If desired number of corners are found in the image then ret = true
#     print(ret)
#     if ret == True:
#         objpoints.append(objp)
#         print("ok1")
#         corners2 = cv2.cornerSubPix(gray, corners, (11,11),(-1,-1), criteria)
#         print("ok2")
#         imgpoints.append(corners2)
#         print(corners2)
#         img = cv2.drawChessboardCorners(img, CHECKERBOARD, corners2, ret)

#     cv2.imshow('img',img)
#     cv2.waitKey(0)

# cv2.destroyAllWindows()
# h,w = img.shape[:2]
imcount = 0
images = glob.glob('./cropped_images/*.png')
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # Find the chess board corners
    # If desired number of corners are found in the image then ret = true
#     img = cv2.imread(fname)
# #     img = cv2.resize(img, (0, 0), fx = 5.0, fy = 5.0)
#     gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Color-segmentation to get binary mask
    lwr = np.array([0, 0, 143])
    upr = np.array([179, 61, 252])
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    msk = cv2.inRange(hsv, lwr, upr)

    # Extract chess-board
    krn = cv2.getStructuringElement(cv2.MORPH_RECT, (50, 30))
    dlt = cv2.dilate(msk, krn, iterations=5)
    res = 255 - cv2.bitwise_and(dlt, msk)

    # Displaying chess-board features
    res = np.uint8(res)
#     cv2.imshow('res', res)
#     cv2.imshow('img',img)
#     cv2.waitKey(0)

#     cv2.destroyAllWindows()

    ret, corners = cv2.findChessboardCorners(res, CHECKERBOARD, cv2.CALIB_CB_ADAPTIVE_

    print(ret)
    """
    If desired number of corner are detected,
    we refine the pixel coordinates and display
    them on the images of checker board
    """
    if ret == True:
        objpoints.append(objp)
        # refining pixel coordinates for given 2d points.
        corners2 = cv2.cornerSubPix(res, corners, (11,11),(-1,-1), criteria)

        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, CHECKERBOARD, corners2, ret)

        cv2.imwrite("./det_images/det_"+str(imcount)+".png", img)
        imcount = imcount+1
#         cv2.imshow('img', img)
#         cv2.waitKey(0)

#         cv2.destroyAllWindows()

#         h,w = img.shape[:2]

        """
        Performing camera calibration by
        passing the value of known 3D points (objpoints)
        and corresponding pixel coordinates of the
        detected corners (imgpoints)
        """
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, ref_gray.shap

print(" ")
print("Recalibration error : \n")
print(ret)
print("Camera matrix : \n")
print(mtx)
print("dist : \n")
print(dist)
print("rvecs : \n")
print(rvecs)
print("tvecs : \n")
print(tvecs)
```

```
True
True
True
True
False
False
False
True
True
False
True
True
True
True
False
True
True
False
True
False

Recalibration error :

1.456337272831923
Camera matrix :

[[1.19659709e+04 0.00000000e+00 2.18885458e+03]
 [0.00000000e+00 1.11169865e+04 2.27818132e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
dist :

[[ 5.66509445e-01 -1.35911355e+01  1.74949766e-03 -4.17521855e-04
   8.90801486e+01]]
rvecs :

(array([[-0.34554271],
       [-0.79370422],
       [ 1.39537772]]), array([[-0.26019112],
       [-0.48122086],
       [ 1.4778129 ]]), array([[0.42503979],
       [0.17218075],
       [1.55988853]]), array([[0.39848043],
       [0.07132641],
       [1.57438533]]), array([[ 0.49074209],
       [-0.00353479],
       [ 1.53293789]]), array([[-0.23026542],
       [-0.61073798],
       [ 1.45788851]]), array([[0.24561736],
       [0.09972415],
       [1.56138936]]), array([[ 0.14837999],
       [-0.01419851],
       [ 1.57015894]]), array([[-0.42317851],
       [-0.46792617],
       [ 1.4093043 ]]), array([[-0.29125655],
       [-0.52029376],
       [ 1.47994626]]), array([[ 0.08991041],
       [-0.64392695],
       [ 1.49631675]]), array([[-0.34433928],
       [-0.64226779],
       [ 1.4559154 ]]), array([[ 0.04197756],
       [-0.57580798],
       [ 1.47441138]]))
tvecs :

(array([[-30.12556031],
       [-45.21071131],
       [232.07709884]]), array([[-29.97073873],
       [-43.2846927 ],
       [232.33810377]]), array([[-29.00052547],
       [-41.08894276],
       [212.73669025]]), array([[-28.93706304],
       [-42.20434553],
       [222.78650969]]), array([[-26.76654438],
       [-39.7949572 ],
       [203.235414  ]]), array([[-30.4353646 ],
       [-43.15545234],
       [224.34737094]]), array([[-30.39115552],
       [-42.82669374],
       [225.81794964]]), array([[-31.95445201],
       [-44.43550829],
       [236.27606437]]), array([[-32.74040317],
       [-46.23818225],
       [239.02550868]]), array([[-28.32732485],
       [-42.00979006],
       [220.42247936]]), array([[-33.37092856],
       [-47.30743796],
       [243.96188555]]), array([[-28.51375419],
       [-41.9516247 ],
       [216.23406105]]), array([[-27.93885891],
       [-42.23350137],
       [215.91581596]]))
```

In [8]:

```
#undistortion
img = cv2.imread('./cb_grids/images/grids_5by4/02.jpg')
h,  w = img.shape[:2]
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))

# undistort
mapx, mapy = cv2.initUndistortRectifyMap(mtx, dist, None, newcameramtx, (w,h), 5)
dst = cv2.remap(img, mapx, mapy, cv2.INTER_LINEAR)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv2.imwrite('calibresult.png', dst)

#reprojection error
mean_error = 0
for i in range(len(objpoints)):
    imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv2.norm(imgpoints[i], imgpoints2, cv2.NORM_L2)/len(imgpoints2)
    mean_error += error
print( "total error: {}".format(mean_error/len(objpoints)) )
```

```
total error: 0.1722079739274847
```

In [ ]: