Assignment_2___Q1___Solution

April 12, 2023

1 Khushdev Pandit

2 Roll no: 2020211

3 *Assignment Question-1*

4

5 Q1 Part-1

-

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
import glob
import matplotlib as mpl
from tqdm import tqdm
mpl.rcParams['figure.facecolor'] = 'white'

# load all the images
images = glob.glob('Chess_Images/*.jpg')
print("Number of Chessboard Images clicked : ", len(images))
```

```
Number of Chessboard Images clicked :  28
```

[ ]:
```
# define the size of the checkerboard
checkerboard_size=(5,5)

# define the world coordinates of the checkerboard
objects_points = []
image_points = []

# termination criteria for the sub-pixel refinement
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
```

```
# loop over all the images to find the chessboard corners
for image_name in tqdm(images):
    img = cv2.imread(image_name)
    inverted_img = np.array(255 - img, dtype=np.uint8)
    gray_img = cv2.cvtColor(inverted_img, cv2.COLOR_BGR2GRAY)

    # 3D points in real world space
    objp = np.zeros((1, checkerboard_size[0]*checkerboard_size[1], 3), np.
 ↪float32)
    objp[0,:,:2] = np.mgrid[0:checkerboard_size[0], 0:checkerboard_size[1]].T.
 ↪reshape(-1, 2)

    # Find the chessboard corners
    cornersFound, corners = cv2.findChessboardCorners(image=gray_img,␣
 ↪patternSize=checkerboard_size, flags=cv2.CALIB_CB_NORMALIZE_IMAGE + cv2.
 ↪CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK)

    if cornersFound == True:
        # Refine the corners
        corners_refined = cv2.cornerSubPix(gray_img, corners, (11,11), (-1,-1),␣
 ↪criteria)
        objects_points.append(objp)
        image_points.append(corners_refined)

        # Draw the corners on the image
        corners_img = cv2.drawChessboardCorners(img, checkerboard_size,␣
 ↪corners_refined, cornersFound)
        for corner in corners_refined.squeeze():
            coord = (int(corner[0]), int(corner[1]))
            cv2.circle(img=corners_img, center=coord, radius=33, color=(255, 0,␣
 ↪0), thickness=15)
        # plt.imsave("Chess_Corners/" + str(image_name.split('\\')[-1].split('.
 ↪')[0] + '.png'), corners_img)
    else:
        print("Corners not found for image: ", image_name)
```

```
100%|      | 28/28 [00:34<00:00,  1.23s/it]
```

[ ]:
```
# Calibrate the camera for all the given images
img = cv2.imread(images[0])
retVal, cameraInternalMatrix, distCoeffs, rvecs, tvecs = cv2.
 ↪calibrateCamera(objects_points, image_points, img.shape[:2],None,None)

print("Error estimate: \n", retVal, "\n")
```

```
print("Internal Camera matrix : \n", cameraInternalMatrix, "\n")
print("Radial Distortion \n", distCoeffs[0], "\n")
print("Focal length :")
print("fx = ", cameraInternalMatrix[0,0])
print("fy = ", cameraInternalMatrix[1,1])
print("\nPrincipal point :")
print("cx = ", cameraInternalMatrix[0,2])
print("cy = ", cameraInternalMatrix[1,2])
print("\nSkew paramerer:")
print("s = ", cameraInternalMatrix[0,1])
```

```
Error estimate:
 2.9029874512404708

Internal Camera matrix :
 [[3.59083306e+03 0.00000000e+00 2.18587381e+03]
 [0.00000000e+00 3.61682218e+03 1.15224849e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Radial Distortion
 [ 0.35342384 -3.46665081 -0.02122771 -0.01373224  5.72094171]

Focal length :
fx =  3590.833064903618
fy =  3616.8221776162177

Principal point :
cx =  2185.873809980998
cy =  1152.2484917564666

Skew paramerer:
s =  0.0
```

[ ]:
```
np.savez('calibration.npz', mtx=cameraInternalMatrix, dist=distCoeffs,␣
↪rvecs=rvecs, tvecs=tvecs)
```

[ ]:
```
data = np.load('calibration.npz')
cameraInternalMatrix = data['mtx']
distCoeffs = data['dist']
rvecs = data['rvecs']
tvecs = data['tvecs']
```

6 Q1 Part-2

-

[ ]:
```
print("Length of Translation Vector: ", len(tvecs))
print("Length of Rotation Vector: ", len(rvecs))
print("Translation Vector Shape: ", np.array(tvecs).shape)
print("Rotation Vector Shape: ", np.array(rvecs).shape)
```

```
Length of Translation Vector:  28
Length of Rotation Vector:  28
Translation Vector Shape:  (28, 3, 1)
Rotation Vector Shape:  (28, 3, 1)
```

[ ]:
```
for i in range(len(tvecs)):
    tranlation_vector = tvecs[i].squeeze()
    Rotation_matrix, Jacobian_matrix = cv2.Rodrigues(rvecs[i])
    print("Translation Vector for Image-" + str(i+1) + ": ", tranlation_vector)
    print("Rotation Matrix for Image-" + str(i+1) + ": \n", Rotation_matrix)
    print()
```

```
Translation Vector for Image-1:  [-2.08240623 -1.5133837  12.7989665 ]
Rotation Matrix for Image-1:
 [[ 0.99942363 -0.0127346  -0.0314681 ]
 [ 0.0128625   0.9999098   0.00386529]
 [ 0.03141603 -0.00426782  0.99949728]]

Translation Vector for Image-2:  [-1.18808451 -1.2212676  15.41348057]
Rotation Matrix for Image-2:
 [[-0.04698458 -0.88119303  0.47041608]
 [ 0.99726649 -0.01449519  0.07245307]
 [-0.05702637  0.47253437  0.87946533]]

Translation Vector for Image-3:  [-4.01634327 -0.91581462 18.73782882]
Rotation Matrix for Image-3:
 [[ 0.90025277 -0.246029    0.35918614]
 [ 0.31149245  0.94038066 -0.13658942]
 [-0.30416674  0.23484878  0.92321647]]

Translation Vector for Image-4:  [-2.94285076 -2.70809673 15.83170226]
Rotation Matrix for Image-4:
 [[ 0.96448887 -0.19012082  0.18334475]
 [ 0.17959713  0.98106169  0.07254542]
 [-0.1936649  -0.03704106  0.98036823]]

Translation Vector for Image-5:  [-1.18142315 -1.85368307 14.60135143]
Rotation Matrix for Image-5:
```

```
 [[ 0.78279454  0.06913201 -0.61842823]
 [ 0.07997233  0.97439252  0.21015149]
 [ 0.61712004 -0.21396258  0.75722049]]

Translation Vector for Image-6:  [-2.26922214 -2.08390956 15.40785112]
Rotation Matrix for Image-6:
 [[ 0.98411516 -0.00936597 -0.17728406]
 [ 0.06898583  0.9403049   0.33326813]
 [ 0.16357969 -0.3402043   0.92601443]]

Translation Vector for Image-7:  [-3.00198614 -0.40534628 14.17422657]
Rotation Matrix for Image-7:
 [[ 0.99951402 -0.00571218  0.03064475]
 [ 0.01859187  0.89831973 -0.43894875]
 [-0.02502143  0.43930518  0.89798936]]

Translation Vector for Image-8:  [-2.80903779 -0.99535319 15.41692731]
Rotation Matrix for Image-8:
 [[ 0.99932428 -0.0293595  -0.02211355]
 [ 0.03094336  0.99669869  0.07506153]
 [ 0.01983678 -0.07569508  0.99693368]]

Translation Vector for Image-9:  [-1.33014272 -1.04335177 16.27272176]
Rotation Matrix for Image-9:
 [[ 0.13509392 -0.98017707  0.14492258]
 [ 0.99063782  0.13071328 -0.03937951]
 [ 0.01965558  0.14888572  0.98865904]]

Translation Vector for Image-10:  [-2.18274857 -0.8480398  13.59361653]
Rotation Matrix for Image-10:
 [[ 0.60143024 -0.56829662  0.56153417]
 [ 0.79485732  0.35479642 -0.49226145]
 [ 0.08052021  0.74240047  0.66510002]]

Translation Vector for Image-11:  [-3.61102746 -0.85718527 17.2646788 ]
Rotation Matrix for Image-11:
 [[ 0.69448285  0.18903612  0.69423261]
 [ 0.10612769  0.92740289 -0.35869318]
 [-0.71163929  0.32278357  0.62400343]]

Translation Vector for Image-12:  [-3.06000649 -0.06465448 13.22776733]
Rotation Matrix for Image-12:
 [[ 0.98904106 -0.09324901 -0.11446573]
 [ 0.03880495  0.91222557 -0.4078464 ]
 [ 0.14244984  0.398935    0.90584707]]

Translation Vector for Image-13:  [ 2.61093154 -1.70283642 12.72729523]
Rotation Matrix for Image-13:
```

```
 [[ 0.05316619 -0.99849253  0.0136394 ]
 [ 0.84155022  0.05215393  0.53765527]
 [-0.53755612 -0.01710684  0.84305443]]

Translation Vector for Image-14:  [ 2.01771979 -0.39455758 13.40439045]
Rotation Matrix for Image-14:
 [[ 0.0874435  -0.99036233  0.10740615]
 [ 0.89782967  0.03164103 -0.43920466]
 [ 0.43157332  0.13483802  0.89194349]]

Translation Vector for Image-15:  [ 1.54976426 -1.0859669  13.78055228]
Rotation Matrix for Image-15:
 [[ 0.03051352 -0.99528873  0.09202865]
 [ 0.96392938  0.00494624 -0.26611218]
 [ 0.26440325  0.09682914  0.95953897]]

Translation Vector for Image-16:  [ 1.6696009   2.50850436 13.59934524]
Rotation Matrix for Image-16:
 [[-9.98941029e-01  2.02246235e-03  4.59644413e-02]
 [-7.84372587e-04 -9.99636801e-01  2.69379068e-02]
 [ 4.60022280e-02  2.68733271e-02  9.98579801e-01]]

Translation Vector for Image-17:  [ 1.97930774 -1.829155   14.02927855]
Rotation Matrix for Image-17:
 [[ 0.00129774 -0.99786131  0.06535384]
 [ 0.94995588  0.02164552  0.31163328]
 [-0.31238141  0.06167885  0.94795231]]

Translation Vector for Image-18:  [-1.90251877 -1.70660719 13.56145013]
Rotation Matrix for Image-18:
 [[ 9.99460910e-01 -3.28311935e-02 -4.53974186e-05]
 [ 2.92303051e-02  8.89209845e-01  4.56564826e-01]
 [-1.49492003e-02 -4.56320023e-01  8.89690147e-01]]

Translation Vector for Image-19:  [-2.54059851 -1.90134045 13.11562995]
Rotation Matrix for Image-19:
 [[ 0.999799   -0.00982866 -0.01747457]
 [ 0.01812687  0.81554176  0.57841426]
 [ 0.0085662  -0.57861475  0.815556  ]]

Translation Vector for Image-20:  [-4.04245872 -0.81103792 15.55140193]
Rotation Matrix for Image-20:
 [[ 0.8757196   0.06724137  0.47811481]
 [ 0.17370992  0.88006483 -0.44193977]
 [-0.45048867  0.47006861  0.75900953]]

Translation Vector for Image-21:  [-4.6585064  -1.32905591 15.66307676]
Rotation Matrix for Image-21:
```

```
 [[ 0.87528898  0.07605092  0.47758294]
 [ 0.0841829   0.94851846 -0.30532929]
 [-0.47621681  0.30745568  0.82382556]]

Translation Vector for Image-22:  [-1.17357333 -1.45005406 13.83622145]
Rotation Matrix for Image-22:
 [[ 0.14137725 -0.87071797  0.47102303]
 [ 0.98919733  0.1056318  -0.10163938]
 [ 0.03874422  0.48030422  0.87624582]]

Translation Vector for Image-23:  [-0.42298795 -1.75963844 13.80001721]
Rotation Matrix for Image-23:
 [[ 0.06184254 -0.84522904  0.53081388]
 [ 0.964065    0.18825821  0.18745004]
 [-0.25836829  0.50014669  0.8264981 ]]

Translation Vector for Image-24:  [-4.92066041 -0.54787213 12.28365138]
Rotation Matrix for Image-24:
 [[ 0.23330213  0.02767523 -0.97201039]
 [ 0.03105504  0.99887297  0.0358939 ]
 [ 0.97190828 -0.03855994  0.23217973]]

Translation Vector for Image-25:  [-2.89373069 -2.55679618 16.73966735]
Rotation Matrix for Image-25:
 [[ 0.38396521 -0.3587514  -0.85080441]
 [ 0.92128765  0.21036512  0.32707121]
 [ 0.06164232 -0.90941957  0.41128612]]

Translation Vector for Image-26:  [-3.46120715  0.48237108 10.9236405 ]
Rotation Matrix for Image-26:
 [[ 0.43305051 -0.06190003 -0.8992417 ]
 [ 0.2314976   0.97181316  0.04458744]
 [ 0.87113496 -0.22748091  0.43517389]]

Translation Vector for Image-27:  [-3.15804519 -0.06449233 11.48591083]
Rotation Matrix for Image-27:
 [[ 0.35721483 -0.21893145 -0.90800142]
 [ 0.47708803  0.87852408 -0.02413392]
 [ 0.80298479 -0.42457562  0.4182714 ]]

Translation Vector for Image-28:  [ 1.52239066 -2.54777009 14.57942674]
Rotation Matrix for Image-28:
 [[ 0.51189856 -0.7769883  -0.36640013]
 [ 0.81073471  0.29594411  0.50510031]
 [-0.28402307 -0.55561342  0.78142474]]
```

7 Q1 Part-3

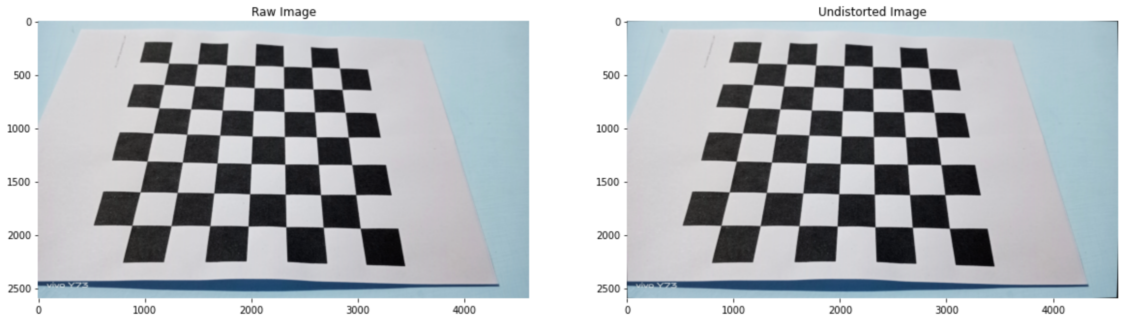
-

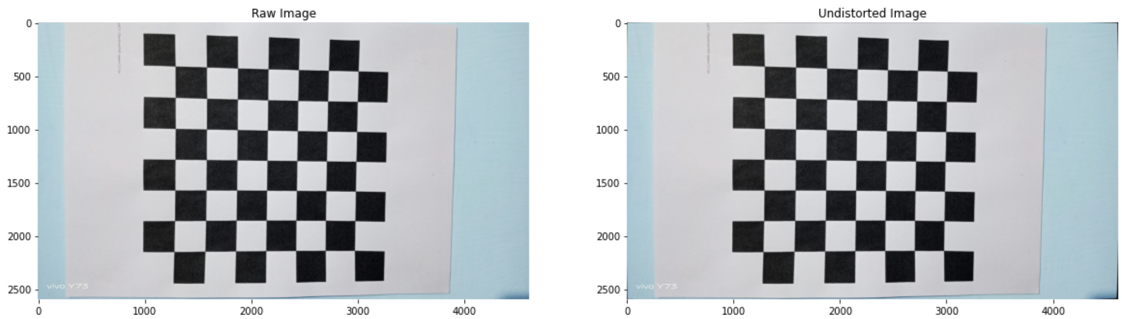
[ ]:
```
print("Lens distortion coefficients :")
print("k1 = ", distCoeffs[0,0])
print("k2 = ", distCoeffs[0,1])
print("p1 = ", distCoeffs[0,2])
print("p2 = ", distCoeffs[0,3])
print("k3 = ", distCoeffs[0,4])
```

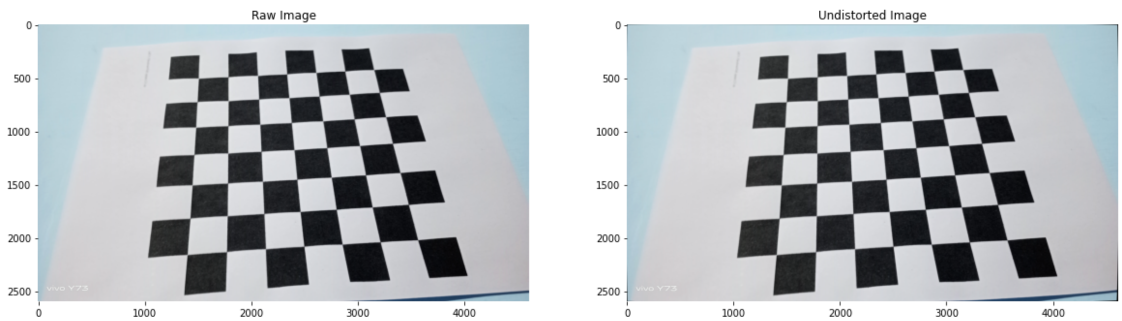
```
Lens distortion coefficients :
k1 =  0.35342384286946465
k2 =  -3.466650807348555
p1 =  -0.02122771187422184
p2 =  -0.013732237993644918
k3 =  5.720941706225883
```

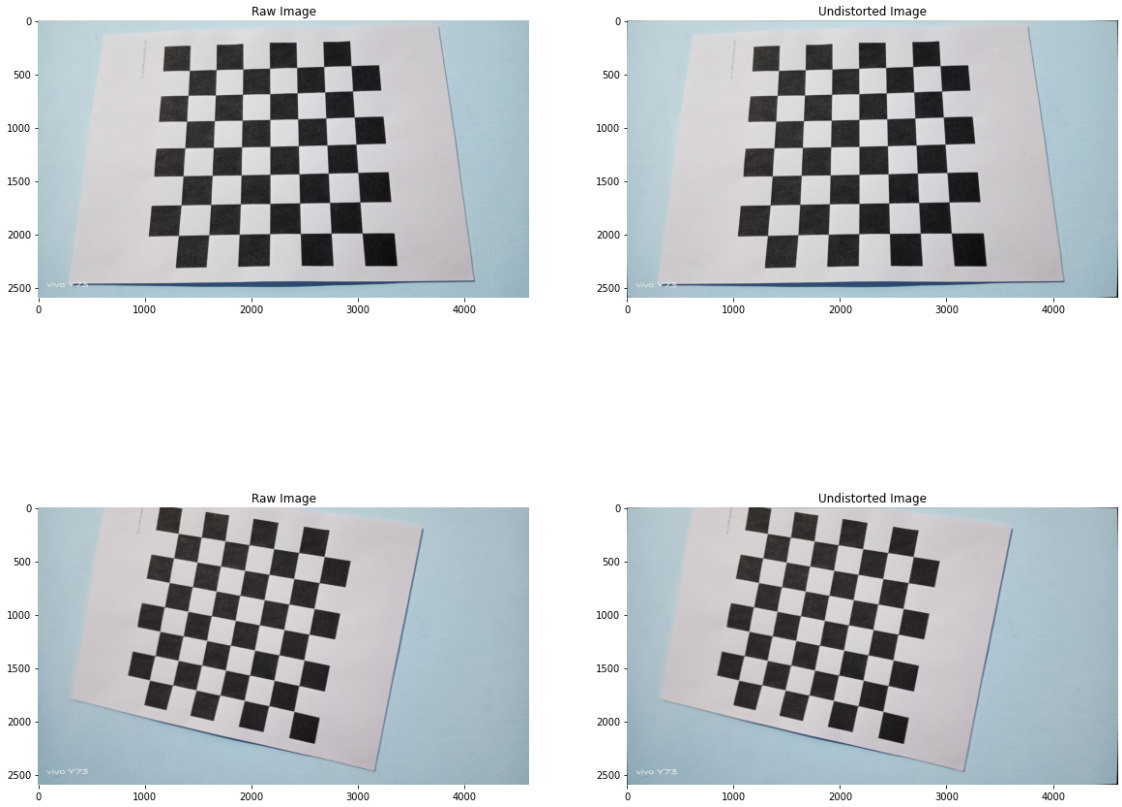
[ ]:
```
images_5 = ['Chess_Images/IMG_20230402_131656.jpg', 'Chess_Images/
 ↪IMG_20230402_131411.jpg',
            'Chess_Images/IMG_20230402_131614.jpg', 'Chess_Images/
 ↪IMG_20230402_131648.jpg',
            'Chess_Images/IMG_20230402_131426.jpg']

for image in images_5:
    img = cv2.imread(image)
    # perform the undistortion operation on the image
    undistorted = cv2.undistort(img, cameraInternalMatrix, distCoeffs)
    # Save the original and undistorted images
    # plt.imsave("Undistorted_Image/Raw_image__" + str(image.split('\\')[-1].
 ↪split('.')[0] + '.png'), img)
    # plt.imsave("Undistorted_Image/Undistorted_image__" + str(image.
 ↪split('\\')[-1].split('.')[0] + '.png'), undistorted)

    fig, ax = plt.subplots(1, 2, figsize=(20, 10))
    ax[0].imshow(img)
    ax[0].set_title("Raw Image")
    ax[0].set_aspect('equal')
    ax[0].set_facecolor('white')
    ax[0].grid(False)
    ax[0].set_frame_on(False)
    ax[1].imshow(undistorted)
    ax[1].set_title("Undistorted Image")
    ax[1].set_aspect('equal')
    ax[1].set_facecolor('white')
```

```
ax[1].grid(False)
ax[1].set_frame_on(False)
plt.show()
```









Observation for the straight lines at the corner of the images change upon application of the distortion coefficients: Straight lines at the corners of the image become somewhat more straight and less distorted on applying the radial distortion coefficients. Radial distortion is caused by the the curvature of the lens. When we perform undistortion of an image (such as for pincushion distortion), we distortion is corrected and a more accurate image scene is obtained.

8 Q1 Part-4

-

[ ]:
```
print("Images Processed:")
errors = []
i = 0
for image_name in tqdm(images):
    # project 3D points to image plane
    reproj_image_points, _ = cv2.projectPoints(objects_points[i], rvecs[i],␣
 ↪tvecs[i], cameraInternalMatrix, distCoeffs)

    # calculate error
```

```
    error = cv2.norm(image_points[i], reproj_image_points, cv2.NORM_L2)/
↪len(reproj_image_points)
    errors.append(error)
    i += 1
```

```
Images Processed:


100%|      | 28/28 [00:00<00:00, 5297.75it/s]
```

[ ]:
```
plt.bar(np.arange(1, len(errors) + 1), errors, 0.8, color='r')
plt.title('Re-projection errors for each image')
plt.xlabel('Image number ')
plt.ylabel('Re-projection error for each image')
plt.xlim([0, len(images)+1])
plt.show()
```

[ ]:
```
# calculate mean and standard deviation of re-projection error
mean_error = np.mean(errors)
std_dev_error = np.std(errors)
```

```
print('Mean Re-projection error for all Images:', mean_error)
print('Standard deviation of Re-projection error for all Images:',␣
 ↪std_dev_error)
```

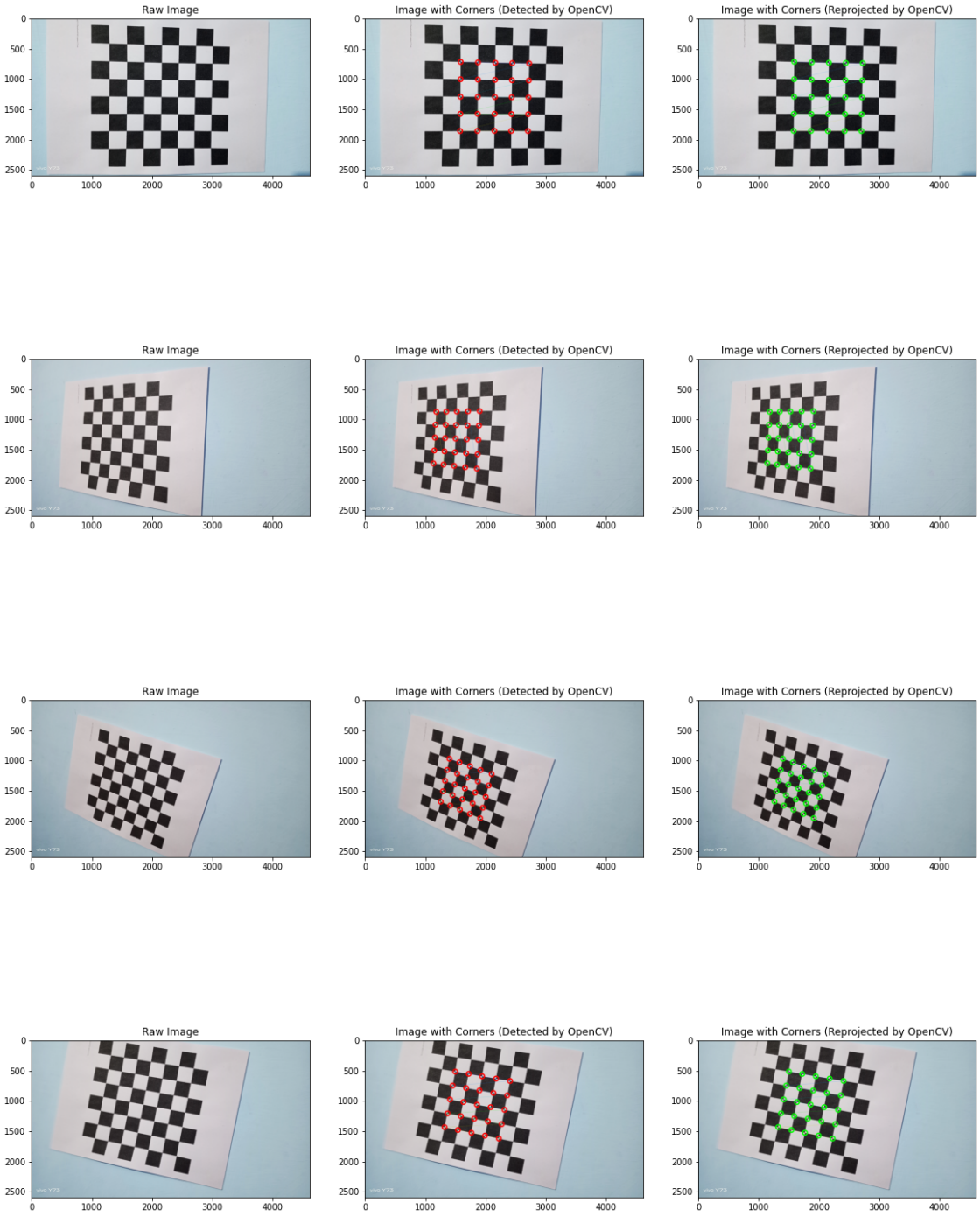
```
Mean Re-projection error for all Images: 0.48873697481323536
Standard deviation of Re-projection error for all Images: 0.31341515953042837
```

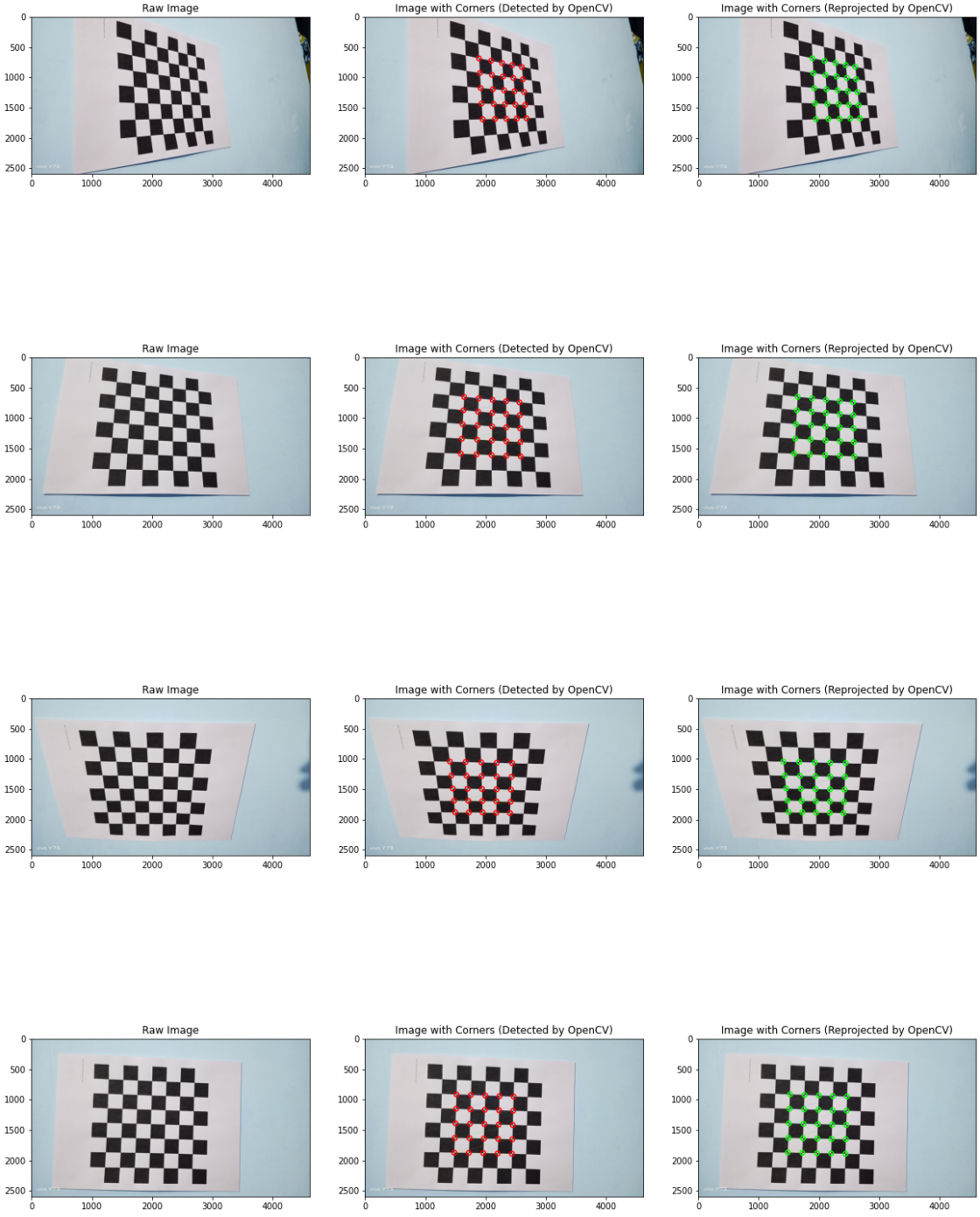
9 Q1 Part-5

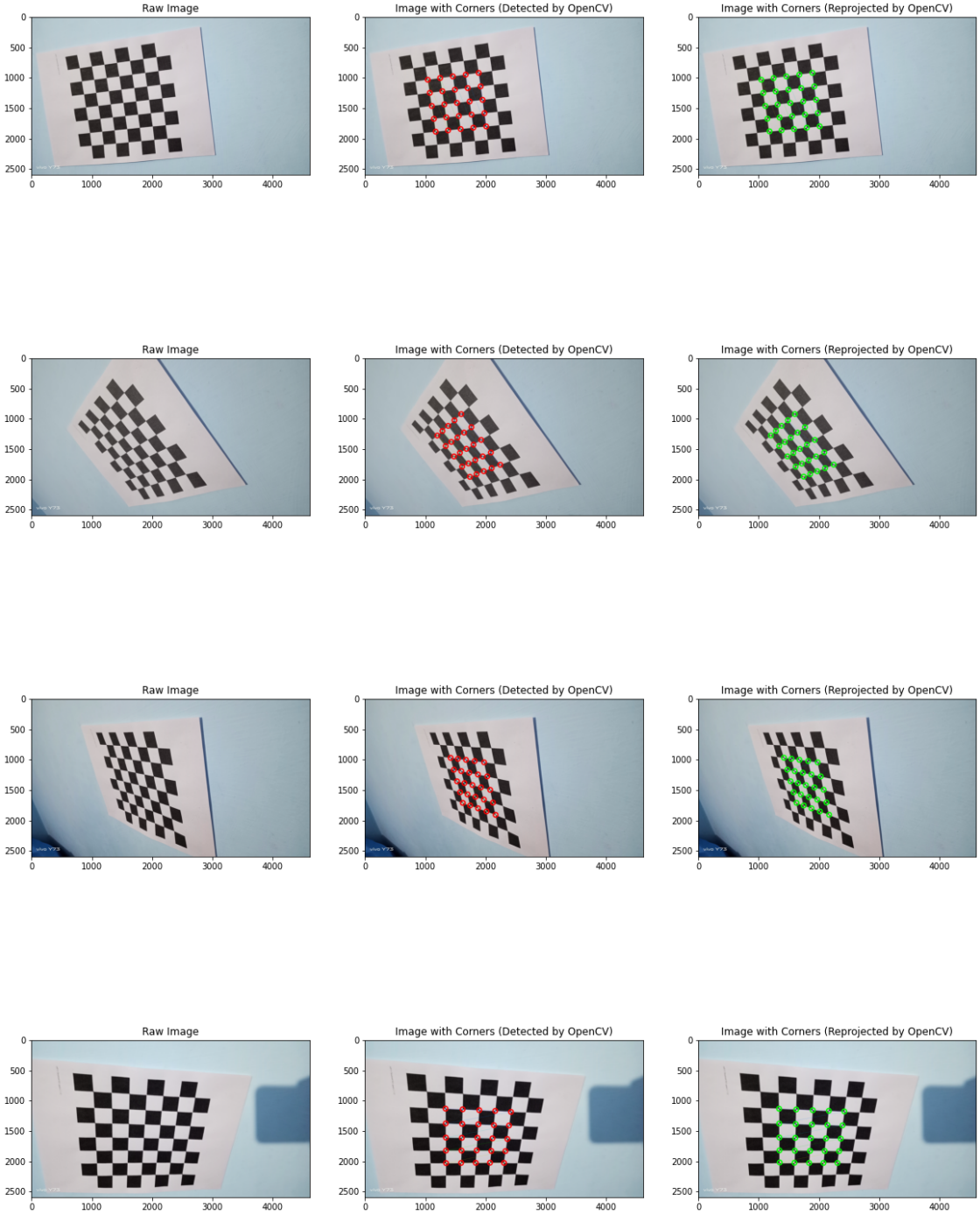
-

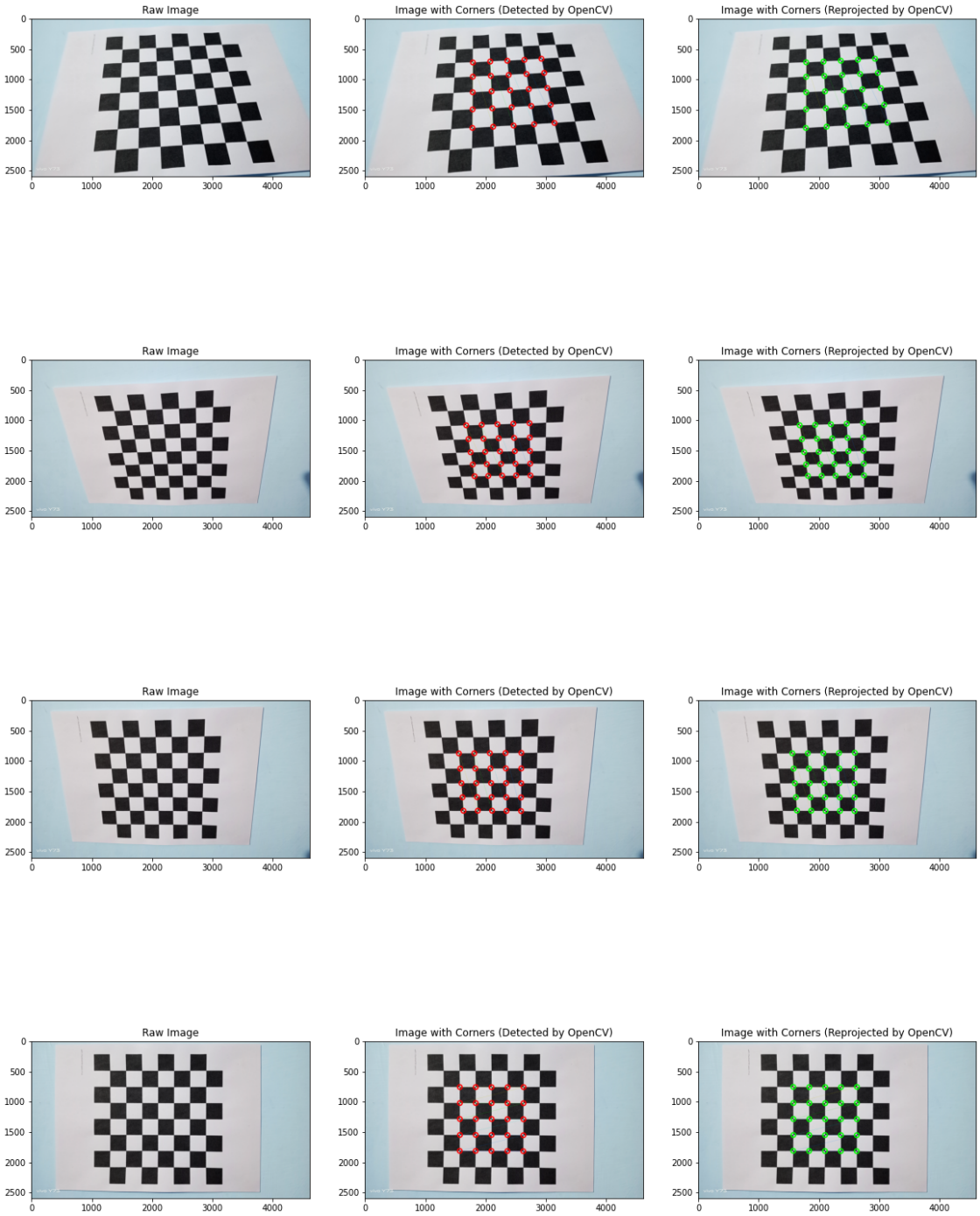
[ ]:
```
i = 0
# Find the Re-projection corners coordinates for all the images
for image_name in images:
    img = cv2.imread(image_name)
    reproj_image_points, _ = cv2.projectPoints(objects_points[i],␣
 ↪rvec=rvecs[i], tvec=tvecs[i],
                                   cameraMatrix=cameraInternalMatrix,␣
 ↪distCoeffs=distCoeffs)

    # Draw the corners on the image
    corners_img = cv2.drawChessboardCorners(img.copy(), checkerboard_size,␣
 ↪image_points[i], True)
    for corner in image_points[i].squeeze():
        coord = (int(corner[0]), int(corner[1]))
        cv2.circle(img=corners_img, center=coord, radius=38, color=(255, 0, 0),␣
 ↪thickness=20)

    # Draw the re-projection corners on the image
    reprojected_img = cv2.drawChessboardCorners(img.copy(), checkerboard_size,␣
 ↪reproj_image_points, True)
    for corner in reproj_image_points.squeeze():
        coord = (int(corner[0]), int(corner[1]))
        cv2.circle(img=reprojected_img, center=coord, radius=38, color=(0, 255,␣
 ↪0), thickness=20)
    i += 1

    # plt.imsave("Chess_Corners_With_ReProjection/Without_Reproj_" +␣
 ↪str(image_name.split('\\')[-1].split('.')[0] + '.png'), corners_img)
    # plt.imsave("Chess_Corners_With_ReProjection/With_Reproj_" +␣
 ↪str(image_name.split('\\')[-1].split('.')[0] + '.png'), reprojected_img)
    fig, ax = plt.subplots(1, 3, figsize=(20, 10))
    ax[0].imshow(img)
```

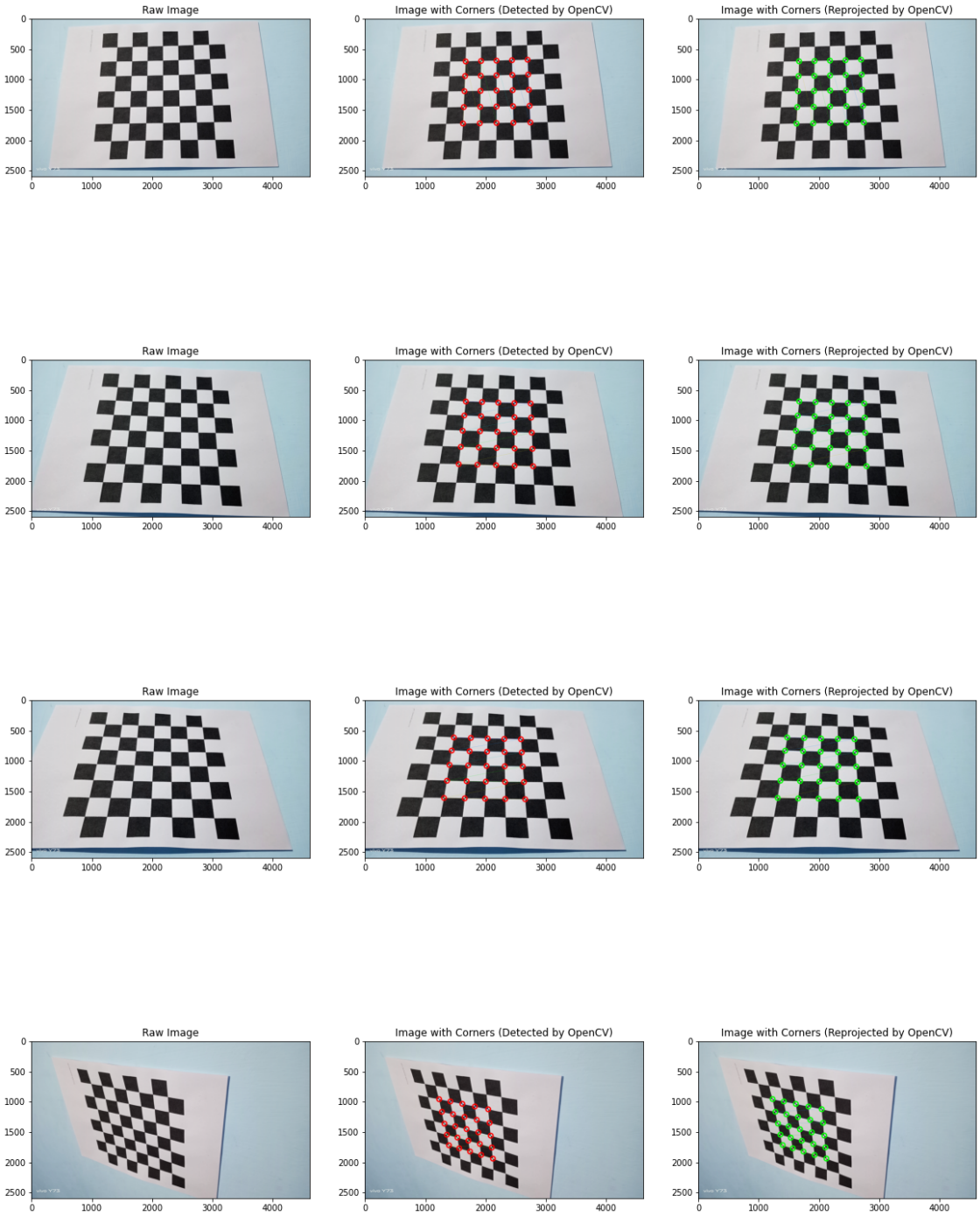
```
    ax[0].set_title("Raw Image")
    ax[1].imshow(corners_img)
    ax[1].set_title("Image with Corners (Detected by OpenCV)")
    ax[2].imshow(reprojected_img)
    ax[2].set_title("Image with Corners (Reprojected by OpenCV)")
    plt.show()
```

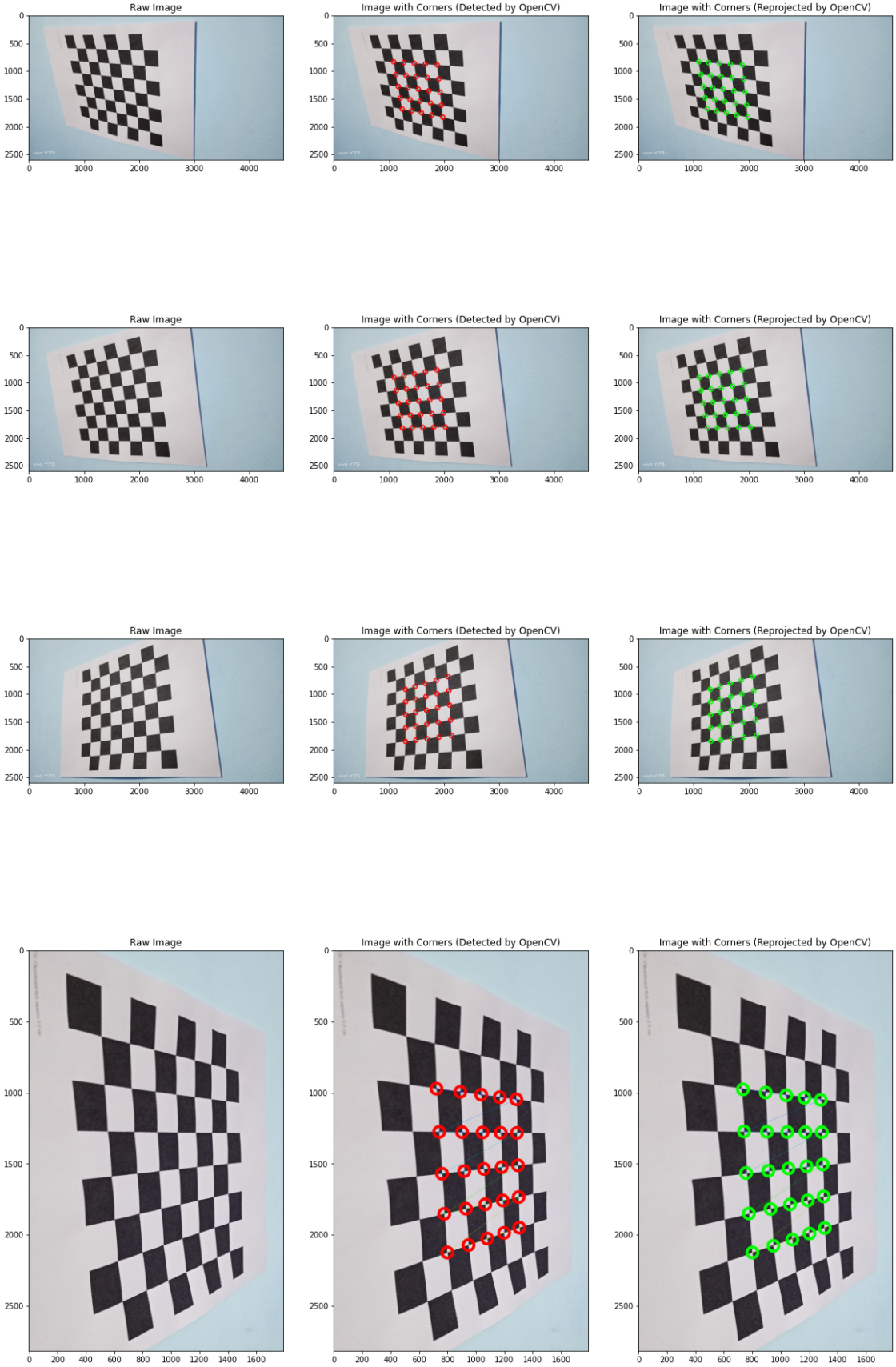


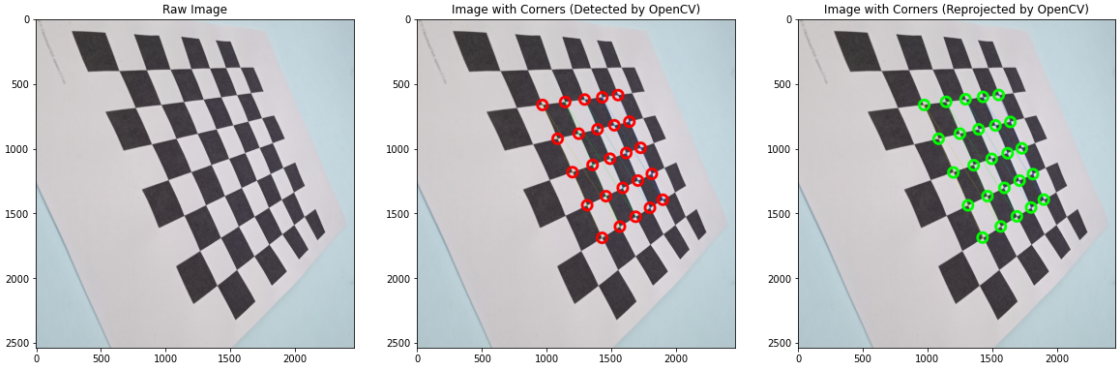

Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)

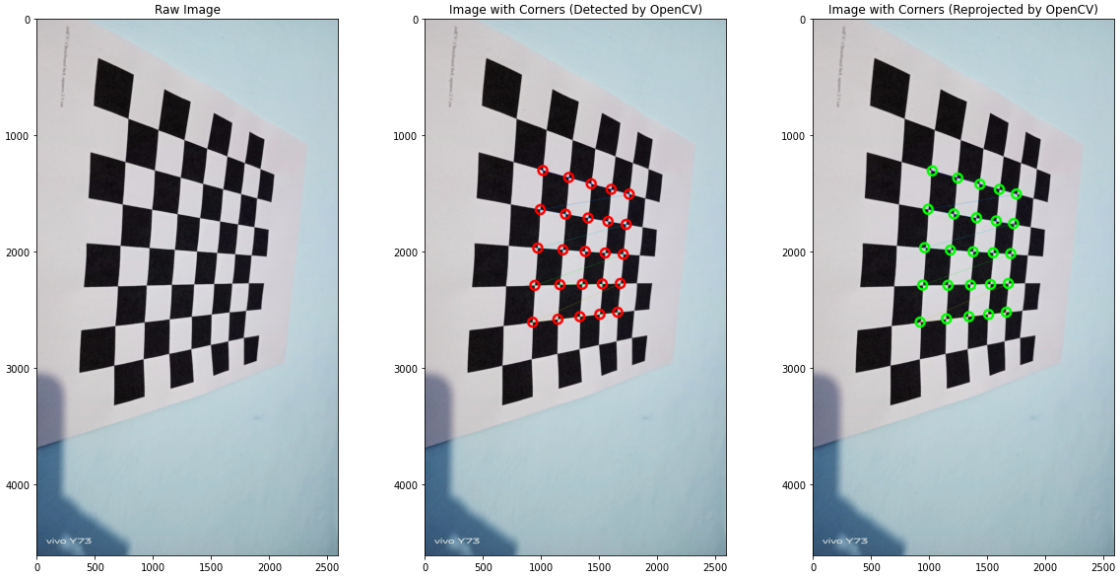

Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)

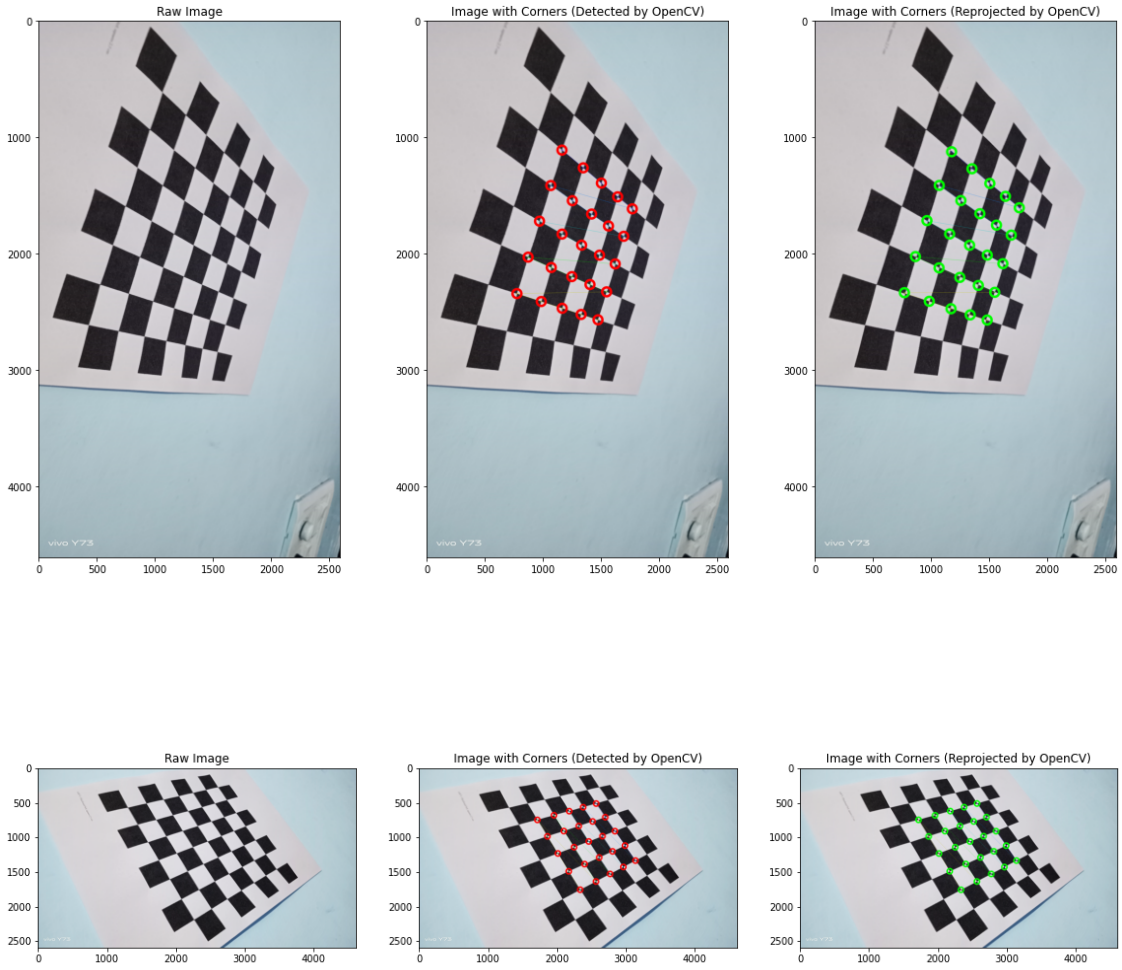

Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)


Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)


Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)


Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)


Raw Image
Image with Corners (Detected by OpenCV)
Image with Corners (Reprojected by OpenCV)
vivo Y73



Comment on how is the reprojection error computed. The Reprojection error between the detected corners and the Re-projected corners is computed by the average of "L2 Norm" error or average "Euclidean distance" error for every image.

10 Q1 Part-6

-

[ ]:
```
# Find the normal vector to the image plane
normal_to_image_planes = []
i = 0

for image_name in images:
    img = cv2.imread(image_name)
```

```
    # Find the normal vector to the image plane
    _, rvec, tvec = cv2.solvePnP(objects_points[i], image_points[i],␣
 ↪cameraInternalMatrix, distCoeffs)

    # Normal vector will be the last column of the rotation matrix
    Rotation_Matrix, _ = cv2.Rodrigues(rvec)
    normal_to_plane = Rotation_Matrix.dot(np.array([0, 0, 1]))
    normal_to_image_planes.append(normal_to_plane)
    i += 1
```

[ ]:
```
# Plotting the normal vectors on the image plane
for i in range(len(images)):
    img = cv2.imread(images[i])
    mean_u = np.median(image_points[i][:,:, 0])
    mean_v = np.median(image_points[i][:,:, 1])

    # calculate z coordinate of the image plane
    z = -(normal_to_image_planes[i][0] * mean_u + normal_to_image_planes[i][1]␣
 ↪* mean_v)  / normal_to_image_planes[i][2]

    # calculate end point of line of the normal vector on the image plane
    end_point = 200*np.array([mean_u, mean_v, z])

    # Drawing the normal vector on the image
    image_with_normal = cv2.line(img, (int(mean_u), int(mean_v)),␣
 ↪(int(end_point[0]), int(end_point[1])), (255, 0, 0), thickness=25,␣
 ↪lineType=8, shift=0)

    if i == 0 or i == 7 or i == 8 or i == 11 or i == 14 or i == 26:
        plt.figure(figsize=(8, 8))
        plt.imshow(image_with_normal)
        plt.title(f"Image-{i} with Normal Vector")
        plt.show()
        # name = images[i].split('\\')[-1].split('.')[0]
        # plt.imsave(f"Image_with_Normal_Vectors/
 ↪Image-{name}_with_Normal_Vector.png", image_with_normal)
```

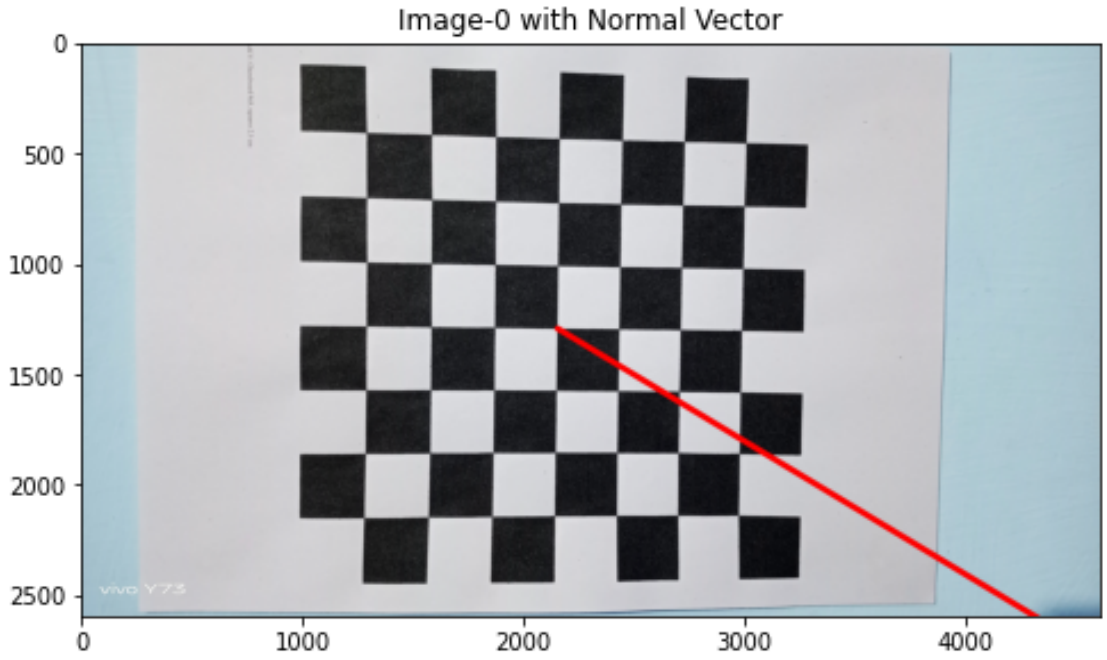


Image-0 with Normal Vector
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000

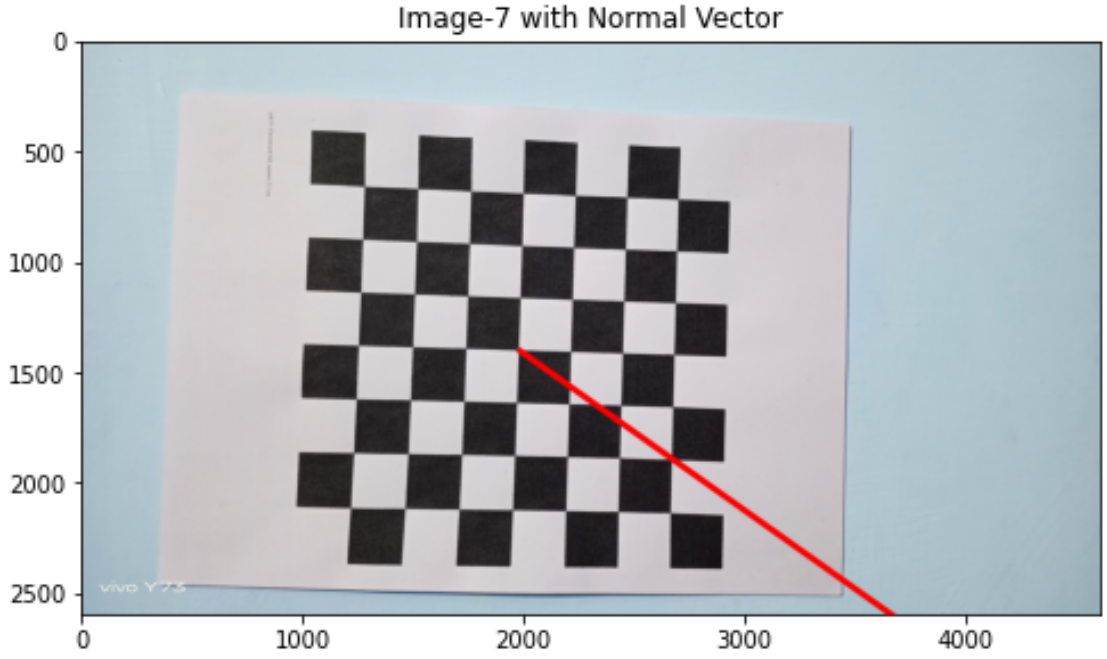


Image-7 with Normal Vector
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000

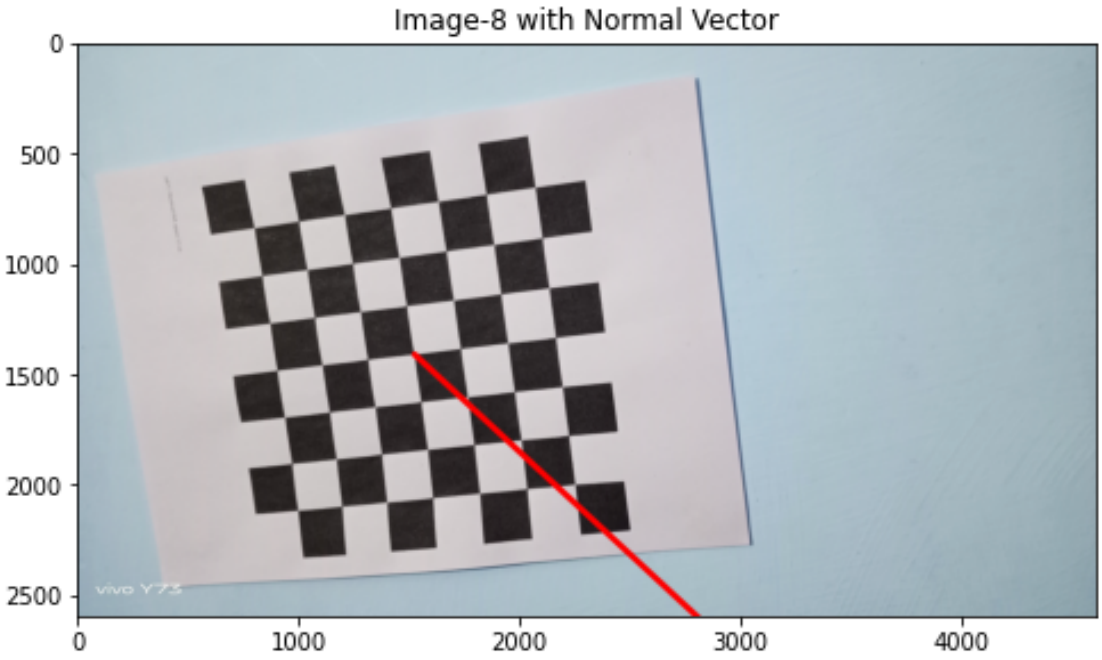


Image-8 with Normal Vector
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000

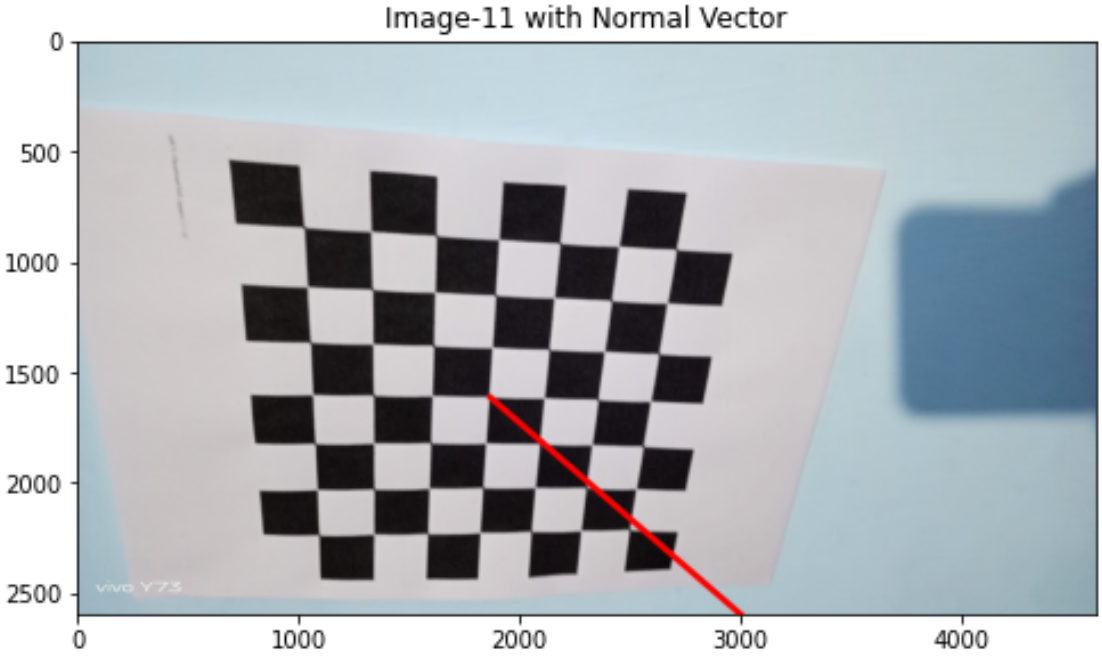


Image-11 with Normal Vector
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000

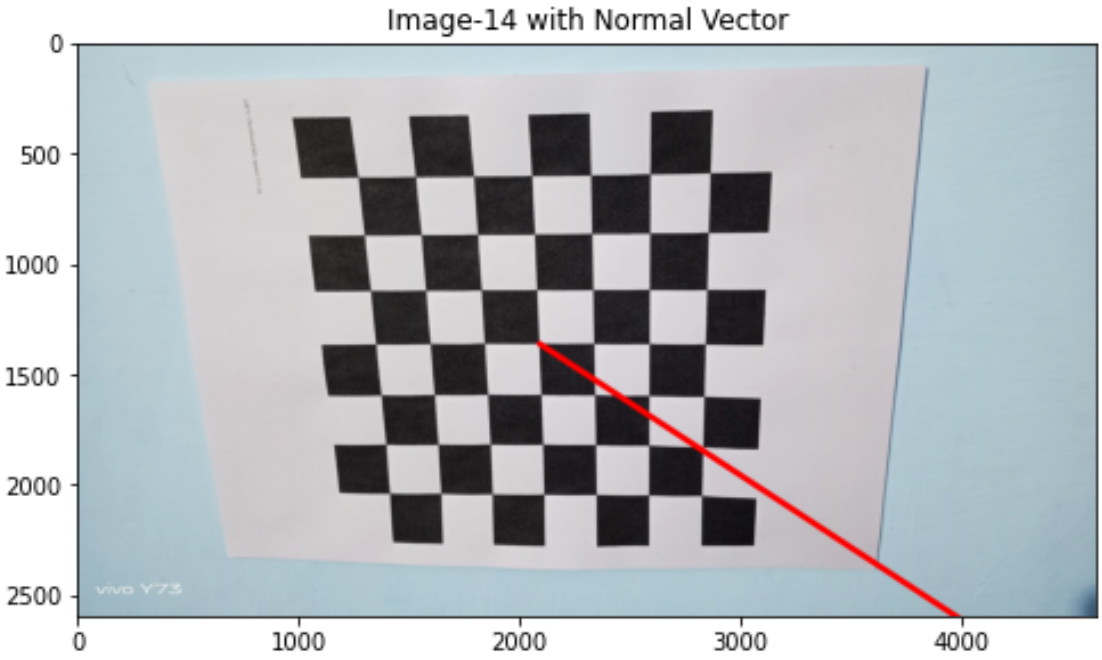
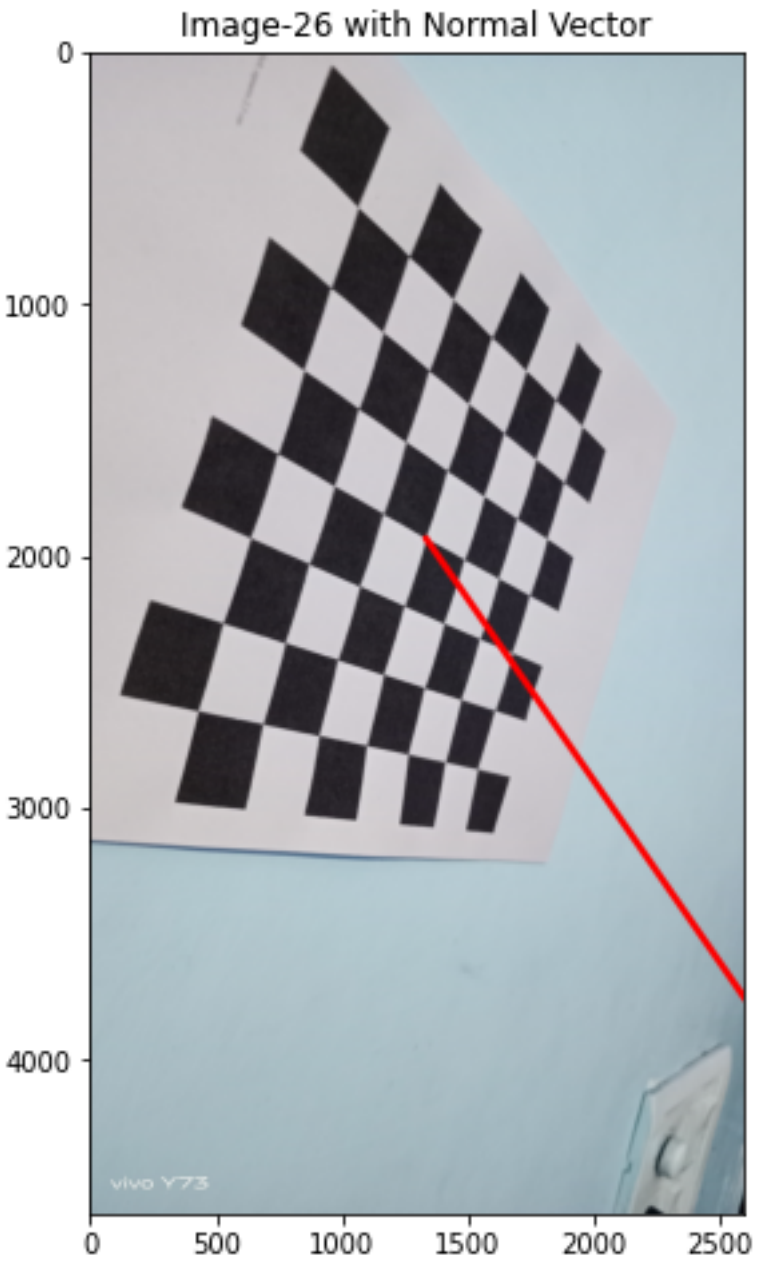


Image-14 with Normal Vector
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000
vivo Y73



[ ]:
```
for i in range(len(normal_to_image_planes)):
    print("Image Number: ", i+1)
    print("Normal Vector to the Image Plane: ", normal_to_image_planes[i], "\n")
```

```
Image Number:  1
Normal Vector to the Image Plane:  [-0.03146808  0.00386531  0.99949728]
```

```
Image Number:  2
Normal Vector to the Image Plane:  [0.47041608 0.07245307 0.87946533]

Image Number:  3
Normal Vector to the Image Plane:  [ 0.35918614 -0.13658943  0.92321647]

Image Number:  4
Normal Vector to the Image Plane:  [0.18334474 0.07254542 0.98036823]

Image Number:  5
Normal Vector to the Image Plane:  [-0.61842823  0.21015149  0.75722049]

Image Number:  6
Normal Vector to the Image Plane:  [-0.17728406  0.33326813  0.92601442]

Image Number:  7
Normal Vector to the Image Plane:  [ 0.03064475 -0.43894875  0.89798936]

Image Number:  8
Normal Vector to the Image Plane:  [-0.02211353  0.07506151  0.99693368]

Image Number:  9
Normal Vector to the Image Plane:  [ 0.14492258 -0.03937949  0.98865904]

Image Number:  10
Normal Vector to the Image Plane:  [ 0.56153417 -0.49226145  0.66510002]

Image Number:  11
Normal Vector to the Image Plane:  [ 0.69423261 -0.35869318  0.62400343]

Image Number:  12
Normal Vector to the Image Plane:  [-0.11446574 -0.4078464   0.90584707]

Image Number:  13
Normal Vector to the Image Plane:  [0.0136394  0.53765527 0.84305443]

Image Number:  14
Normal Vector to the Image Plane:  [ 0.10740615 -0.43920466  0.89194349]

Image Number:  15
Normal Vector to the Image Plane:  [ 0.09202865 -0.26611218  0.95953897]

Image Number:  16
Normal Vector to the Image Plane:  [0.04596443 0.02693791 0.9985798 ]

Image Number:  17
Normal Vector to the Image Plane:  [0.06535384 0.31163328 0.94795231]
```

```
Image Number:  18
Normal Vector to the Image Plane:  [-4.53946442e-05  4.56564828e-01
8.89690146e-01]

Image Number:  19
Normal Vector to the Image Plane:  [-0.01747457  0.57841426  0.815556  ]

Image Number:  20
Normal Vector to the Image Plane:  [ 0.47811481 -0.44193977  0.75900953]

Image Number:  21
Normal Vector to the Image Plane:  [ 0.47758294 -0.30532929  0.82382556]

Image Number:  22
Normal Vector to the Image Plane:  [ 0.47102303 -0.10163938  0.87624582]

Image Number:  23
Normal Vector to the Image Plane:  [0.53081388 0.18745004 0.8264981 ]

Image Number:  24
Normal Vector to the Image Plane:  [-0.97201039  0.0358939   0.23217973]

Image Number:  25
Normal Vector to the Image Plane:  [-0.85080441  0.32707121  0.41128612]

Image Number:  26
Normal Vector to the Image Plane:  [-0.8992417   0.04458744  0.43517389]

Image Number:  27
Normal Vector to the Image Plane:  [-0.90800142 -0.02413392  0.4182714 ]

Image Number:  28
Normal Vector to the Image Plane:  [-0.36640013  0.50510031  0.78142474]
```