Assignment 5 -Group 9

Camera calibration using builtin function.

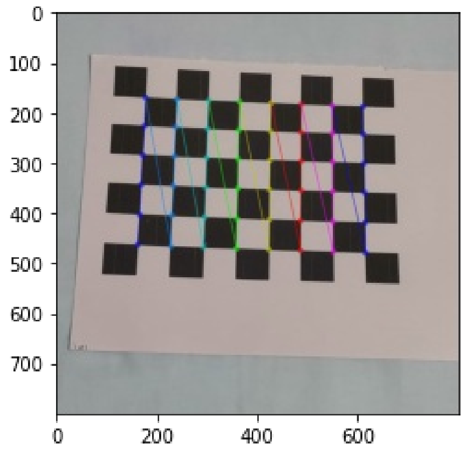
Hint:

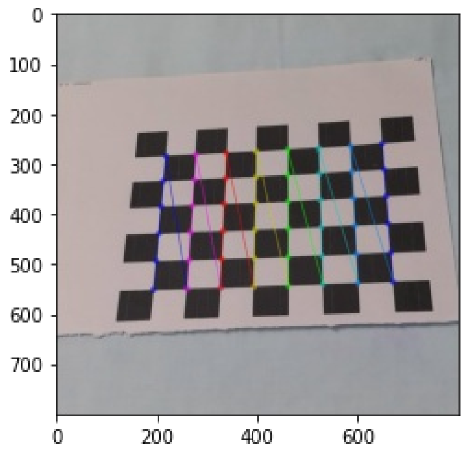
1. **Chessboard can be used for imaging purpose**
2. **All the pre-processing required for the purposes are expected to be done**

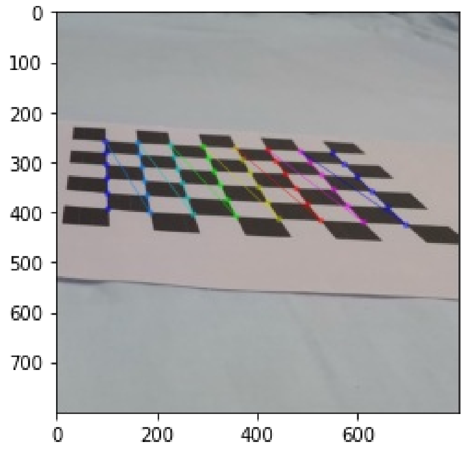
In [3]:

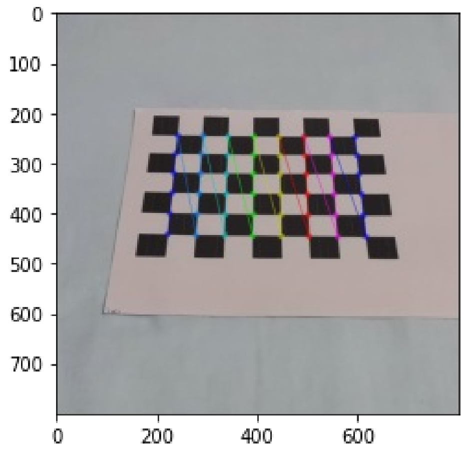
```
import cv2
import numpy as np
import glob
import matplotlib.pyplot as plt


sqr_x = 9 # Number of chessboard squares along the x-axis
sqr_y = 7  # Number of chessboard squares along the y-axis
p_x = sqr_x - 1 # Number of interior corners along x-axis
p_y = sqr_y - 1 # Number of interior corners along y-axis

# Store vectors of 3D points for all chessboard images (world coordinate frame)
pts_obj = []

# Store vectors of 2D points for all chessboard images (camera coordinate frame)
pts_im = []

# Set termination criteria. We stop either when an accuracy is reached or when
# we have finished a certain number of iterations.
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# Define real world coordinates for points in the 3D coordinate frame
# Object points are (0,0,0), (1,0,0), (2,0,0) ...., (5,8,0)
pts_obj_3D = np.zeros((p_x * p_y, 3), np.float32)

# These are the x and y coordinates
pts_obj_3D[:,:2] = np.mgrid[0:p_y, 0:p_x].T.reshape(-1, 2)


igs = glob.glob('input/*.jpeg')

for fle in igs:

    image = cv2.imread(fle)

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Find the corners on the chessboard
    ret, corners = cv2.findChessboardCorners(gray, (p_y, p_x), None)

    if ret == True:


        # Append object points
        pts_obj.append(pts_obj_3D)

        # Find more exact corner pixels
        corners_2 = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        # Append image points
        pts_im.append(corners)

        cv2.drawChessboardCorners(image, (p_y, p_x), corners_2, ret)
        plt.imshow(image)
        plt.show()
```

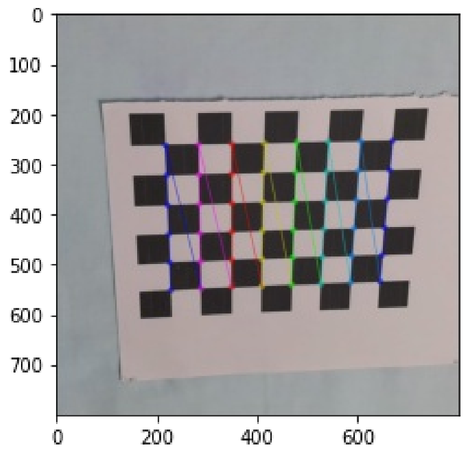
```
distorted_image = cv2.imread('input/qq3.jpeg')

# Perform camera calibration to return the camera matrix, distortion coefficients, rotati
on and translation vectors etc
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(pts_obj,
                                                   pts_im,
                                                   gray.shape[::-1],
                                                   None,
                                                   None)

height, width = distorted_image.shape[:2]

# Returns optimal camera matrix and a rectangular region of interest
optimal_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(mtx, dist,
                                                           (width,height),
                                                           1,
                                                           (width,height))


undistorted_image = cv2.undistort(distorted_image, mtx, dist, None,
                                  optimal_camera_matrix)

# Crop the image.
x, y, w, h = roi
undistorted_image_cut = undistorted_image[y:y+h, x:x+w]


im_ = cv2.hconcat([distorted_image,undistorted_image ])

cv2.imshow('corrected', im_)
cv2.waitKey(0)
cv2.imshow('corrected-cut', undistorted_image_cut)
cv2.waitKey(0)

cv2.destroyAllWindows()

cv2.imwrite('output1.jpg',im_)
cv2.imwrite('output2.jpg',undistorted_image_cut)
```

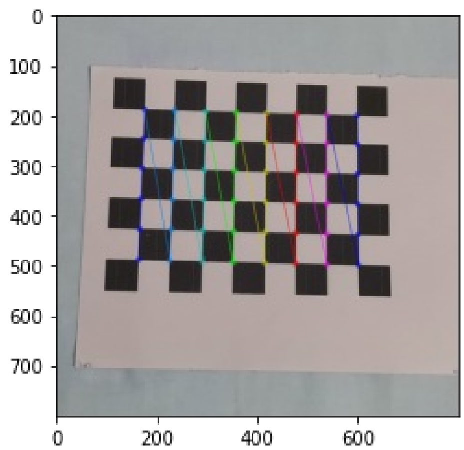


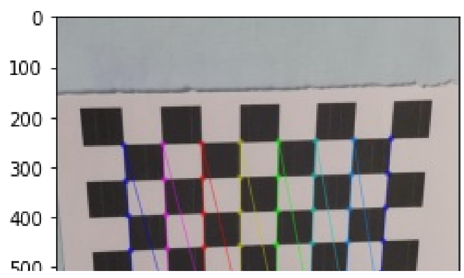


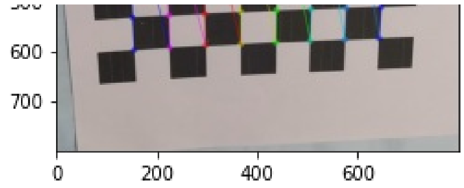

0
100
200
300
400
500
600
700
0 200 400 600

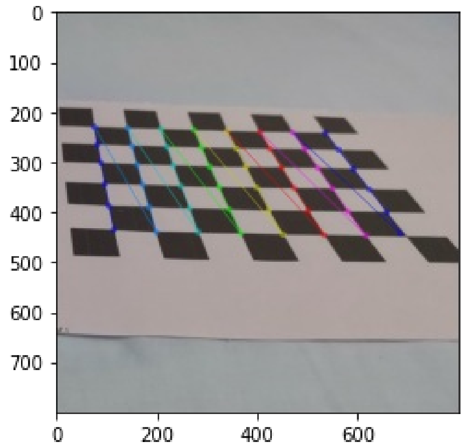

0
100
200
300
400
500
600
700
0 200 400 600

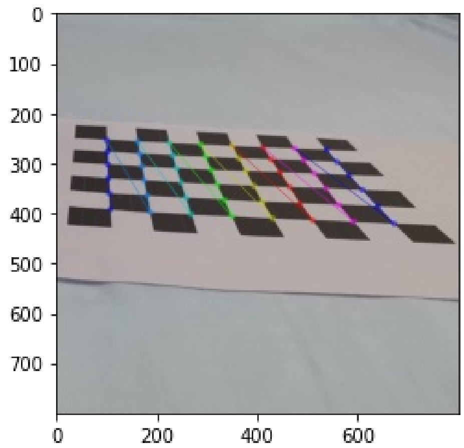

0
100
200
300
400
500
600
700
0 200 400 600

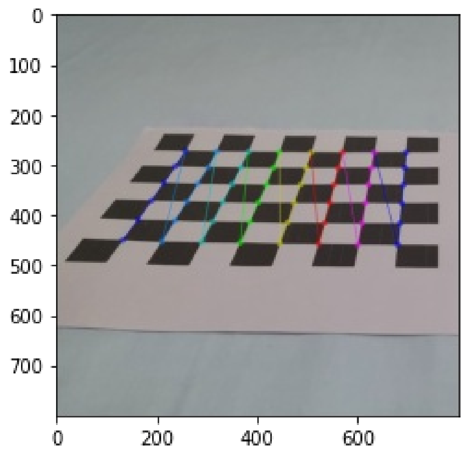

0
100
200
300
400
500
600
700
0 200 400 600

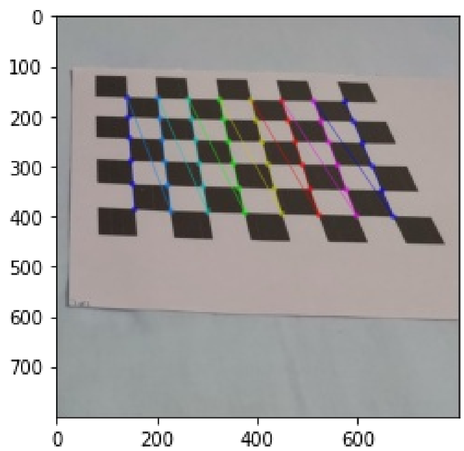

0
100
200
300
400
500


600
700
0
200
400
600

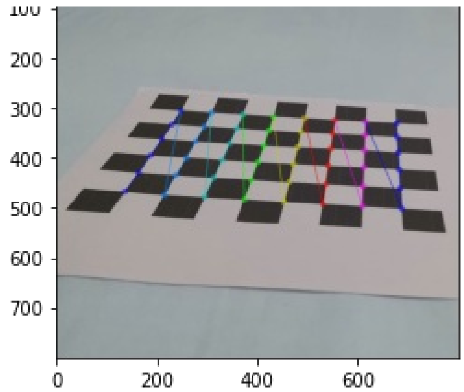

0
100
200
300
400
500
600
700
0
200
400
600


0
100
200
300
400
500
600
700
0
200
400
600


0
100
200
300
400
500
600
700
0
200
400
600


0
100
200
300
400
500
600
700
0
200
400
600


0



Out[3]:

```
True
```

In [5]:

```
print("Optimal Camera matrix:")
print(optimal_camera_matrix)

print(" Distortion coefficient:")
print(dist)

print("Rotation Vectors:")
print(rvecs)

print("Translation Vectors:")
print(tvecs)
```

```
Optimal Camera matrix:
[[1.38696851e+03 0.00000000e+00 4.11763798e+02]
 [0.00000000e+00 1.35804333e+03 3.87121956e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
 Distortion coefficient:
[[ 1.47500679e+00 -5.62412832e+01  7.24056971e-04  2.80107614e-02
   3.98890635e+02]]
Rotation Vectors:
(array([[-0.13105222],
       [-0.29205271],
       [-1.53444873]]), array([[-0.3410365 ],
       [ 0.52239027],
       [ 1.50859122]]), array([[-0.7015745 ],
       [-1.11961335],
       [-1.48069683]]), array([[-0.50879028],
       [-0.53589378],
       [-1.50099316]]), array([[ 0.17553168],
       [-0.37948672],
       [ 1.54404892]]), array([[ 0.0173486 ],
       [-0.19305944],
       [-1.54778254]]), array([[ 0.14395081],
       [-0.36723023],
       [ 1.53942538]]), array([[-0.49576429],
       [-1.04078491],
       [-1.58014807]]), array([[-0.67305459],
       [-1.09682352],
       [-1.49330424]]), array([[-0.93540464],
       [-0.59890566],
       [-1.2829476 ]]), array([[-0.4080085 ],
       [-0.80566182],
       [-1.56703237]]), array([[-0.89562471],
       [-0.73128759],
       [-1.27552584]]))
Translation Vectors:
(array([[-3.68078791],
       [ 1.2414972 ],
       [21.90561671]]), array([[ 3.88968946],
       [-2.03916181],
       [23.40540872]]), array([[-3.36909231],
       [ 0.14709512],
       [15.7598513 ]]), array([[-3.22582883],
```

```
[ 1.05506006],
[25.83140033]]), array([[ 4.12087818],
[-2.13447136],
[22.87668485]]), array([[-3.78445588],
[ 1.669783  ],
[22.6830953 ]]), array([[ 3.4700005 ],
[-1.9615599 ],
[19.42480237]]), array([[-3.16516458],
[ 0.61797441],
[15.23906812]]), array([[-3.49355557],
[ 0.13473153],
[16.57649733]]), array([[-3.4277148 ],
[ 0.85434943],
[19.18765511]]), array([[-3.22539365],
[ 0.0892318 ],
[18.16656545]]), array([[-3.3765964 ],
[ 1.11094292],
[19.20251877]]]))
```

In [ ]:

In [ ]: