# camera_calibration

February 18, 2019

## 0.1 Assignment 1 Camera Calibration

### 0.1.1 Name : Nitin Kandpal, Roll number : 2018802004

```
In [1]: %pylab inline
        import cv2
        import numpy as np
        import random
        from matplotlib import pyplot as plt
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: img_points = np.array([[129,273],[889,321],[1649,369],[2433,417],[3233,465],[4067,513],[
        [193,1041],[971,1097],[1665,1161],[2433,1209],[3217,1273],[4033,1329],[4865,1385],
        [665,2097],[1433,2161],[2225,2233],
        [329,2393],[1137,2457],
        [357,3209],[1409,3305],[2377,3393],[3385,3489],[4417,3609]],dtype='float32')


        world_points = np.array([[216,72,0],[180,72,0],[144,72,0],[108,72,0],[72,72,0],[36,72,0]
                        [216,36,0],[180,36,0],[144,36,0],[108,36,0],[72,36,0],[36,36,0],[0,36,0]
                        [180,0,36],[144,0,36],[108,0,36],
                        [180,0,72],[144,0,72],
                        [144,0,144],[108,0,144],[72,0,144],[36,0,144],[0,0,144]],dtype='float32'
```

### 0.1.2 answer 1 DLT

```
In [3]: def point_world_matrix(img_point, world_point):

            pointX = [world_point[0],world_point[1],world_point[2],1,0,0,0,0,-img_point[0]*world
            pointY = [0,0,0,0,world_point[0],world_point[1],world_point[2],1,-img_point[1]*world
            return pointX,pointY        ## A is 2 x 12 matrix

In [4]: def DLT_calibration(img_points,world_points):
            #M = np.empty((2*img_points.shape[0], 12),dtype='int64')
            A = []
            objpoints = [] # 3d point in real world space
            imgpoints = [] # 2d points in image plane.
```

```
    objpoints_DLT = []
    imgpoints_DLT = []
    count = 0
    for img_point,world_point in zip(img_points,world_points):
        pointX,pointY = point_world_matrix(img_point, world_point)   ## 2x12 matrix

        #np.append(M,np.array(A_point),axis=0)
        A.append(pointX)
        A.append(pointY)
        if(count<12):
            objpoints.append(world_point)
            imgpoints.append(img_point)

        count=count+1
    A = np.array(A)



    ### perfomr SVD
    u, s, vh = np.linalg.svd(A, full_matrices=True)
    ## use the last value
    M = np.transpose(vh)

    M_1d = M[:,-1]
    M_2d = np.reshape(M_1d,[3,4])
    ## to calculate redial distortion
    objpoints = np.array(objpoints)
    objpoints.astype('float32')
    objpoints_DLT.append(objpoints)
    imgpoints = np.array(imgpoints)
    imgpoints.astype('float32')
    imgpoints = np.reshape(imgpoints,[imgpoints.shape[0],1,imgpoints.shape[1]])
    imgpoints_DLT.append(imgpoints)
    return M_2d, objpoints_DLT,imgpoints_DLT
```

```
In [5]: def draw_reproject_point(img,world_points,P):
    img1 = img.copy()
    count = 0
    for world_point in world_points:
        h_world_points = np.array([world_point[0],world_point[1],world_point[2],1])
        img_point = np.dot(P,h_world_points)

        X = int(round(img_point[0]/img_point[2]))
        Y = int(round(img_point[1]/img_point[2]))
        #print(X,Y)
        ### draw points
        cv2.rectangle(img1, (X,Y),(X+10,Y+10),[0,255,0],20)
        if (count > 0):
```

```
                cv2.line(img1, (X_old,Y_old), (X,Y), (0, 255, 0), thickness=4, lineType=8)
            X_old = X
            Y_old = Y
            count =count+1
            #cv2.circle(img, (X,Y), 55,[0,255,0])
        imgplot = plt.imshow(img1)
        plt.show()
```

```
In [6]: def reprojection_error(img_points,world_points,P):
            error = 0
            for img_point, world_point in zip(img_points,world_points):
                h_world_points = np.array([world_point[0],world_point[1],world_point[2],1])
                predicted_img_point = np.dot(P,h_world_points)
                X = int(round(predicted_img_point[0]/predicted_img_point[2]))
                Y = int(round(predicted_img_point[1]/predicted_img_point[2]))
                temp_error = np.linalg.norm(img_point-np.array([X,Y]))
            error = temp_error + error
            return error
```

### 0.1.3 answer2 ransac variation

```
In [7]: def ransac_variation_calibration(img_points,world_points):

            ## select random points
            count =0
            while(True):
                index = random.sample(range(len(img_points)), 12)
                random_img_points = img_points[index]
                random_world_points = world_points[index]
                M1,_,_ = DLT_calibration(random_img_points,random_world_points)

                r_error = reprojection_error(img_points,world_points,M1)
                if(r_error < 6):
                    break
            return M1
            #return r_error
```

### 0.1.4 Answer 3 Use DLT and ransac to calibrate camera

```
In [8]: M_DLT,objpoints,imgpoints = DLT_calibration(img_points,world_points)
        print(M_DLT)
```

```
[[-3.97632887e-03 -3.84684271e-04 -1.93680152e-03  9.11764404e-01]
 [-2.23169425e-04 -4.29677925e-03  7.26016697e-04  4.10666392e-01]
 [ 5.88240497e-08 -1.43347861e-07 -3.31500057e-07  1.90394071e-04]]
```

```
In [9]: M_ransac = ransac_variation_calibration(img_points,world_points)
        print(M_ransac)
```

```
[[ 3.99306319e-03  3.31010723e-04  1.83874926e-03 -9.11714708e-01]
 [ 2.20484107e-04  4.29619699e-03 -8.21407239e-04 -4.10776872e-01]
 [-6.24575276e-08  1.34460228e-07  3.04926363e-07 -1.90348925e-04]]
```

```
In [10]: reprojection_error(img_points,world_points,M_ransac)
```

```
Out[10]: 4.123105625617661
```

Ransac variation improved the projection result due to variation in selecting image points in DLT method. we can see the reprojection matrix are comming very small in some units migh be in mm.

### 0.1.5 Answer 4 distortion coeffcients

```
In [11]: gray = cv2.imread('../IMG_5455.JPG',0)
         gray.shape
         criteria = (cv2.CALIB_USE_INTRINSIC_GUESS + cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_M
```

```
In [12]:
         def rq(A):
             '''Implement rq decomposition using QR decomposition

             From Wikipedia,
              The RQ decomposition transforms a matrix A into the product of an upper triangular
              QR decomposition is Gram-Schmidt orthogonalization of columns of A, started from t
              RQ decomposition is Gram-Schmidt orthogonalization of rows of A, started from the
             '''
             A = np.asarray(A)

             m, n = A.shape

             # Reverse the rows
             reversed_A = np.flipud(A)

             # Make rows into column, then find QR
             Q, R = np.linalg.qr(np.transpose(reversed_A))

             # The returned R is flipped updown, left right of transposed R
             R = np.flipud(np.transpose(R))
             R[:,0:m-1] = R[:,m-1:0:-1]

             # The returned Q is the flipped up-down of transposed Q
             Q = np.transpose(Q)
             Q[0:m-1, :] = Q[m-1:0:-1, :]

             return R, Q
```

In [13]: #K = M[0:3,0:3]

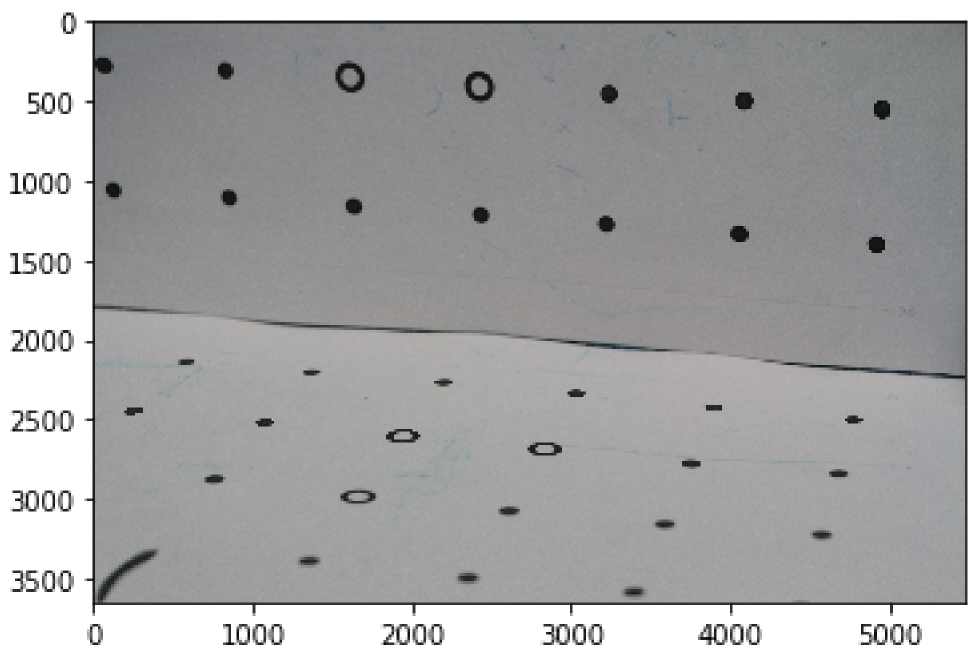
### 0.1.6 find distortion coefficients

```
In [14]: ret_DLT, mtx_DLT, dist_DLT, rvecs_DLT, tvecs_DLT = cv2.calibrateCamera(objpoints,imgpoi
```

```
In [15]: dist_DLT
```

```
Out[15]: array([[-2.07868251e+00,  4.06397153e+01, -1.78786053e-02,
                  7.90737536e-03, -2.17138037e+02]])
```

```
In [16]: test_img = cv2.imread('../IMG_5455.JPG')
         h,  w = test_img.shape[:2]
         #newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx_R,dist_R,(w,h),alpha= 0)
         newMat_R, ROI = cv2.getOptimalNewCameraMatrix(mtx_DLT, dist_DLT,(w,h), alpha = -1, cent
         #mapx, mapy = cv2.initUndistortRectifyMap(mtx_DLT, dist_DLT, None, newMat_R, (w,h), m1t
         #
         #dst = cv2.remap(test_img, mapx, mapy, cv2.INTER_LINEAR)
         dst = cv2.undistort(test_img, mtx_DLT, dist_DLT,newMat_R)
         cv2.imwrite('../undistort.JPG',dst)
         imgplot = plt.imshow(dst)
         plt.show()
```




```
In [17]: ## mnually read new image points in undistorted image
```

```
u_img_points = np.array([[65,273],[825,313],[1609,345],[2417,417],[3233,457],[4081,505]
[121,1057],[849,1113],[1633,1161],[2433,1209],[3217,1273],[4057,1337],[4913,1393],
[585,2129],[1369,2201],[2201,2265],
[249,2449],[1081,2513]],dtype='float32')

u_world_points = world_points[0:-5]
```

```
In [18]: M_DLT_undistorted,_,_ = DLT_calibration(u_img_points,u_world_points)
         print(M_DLT_undistorted)
```

```
[[-4.07209430e-03 -2.70677421e-04 -1.87649578e-03  9.11592672e-01]
 [-2.15093767e-04 -4.32646592e-03  8.65151566e-04  4.11046319e-01]
 [ 6.74766195e-08 -9.85129517e-08 -2.92906725e-07  1.87322371e-04]]
```

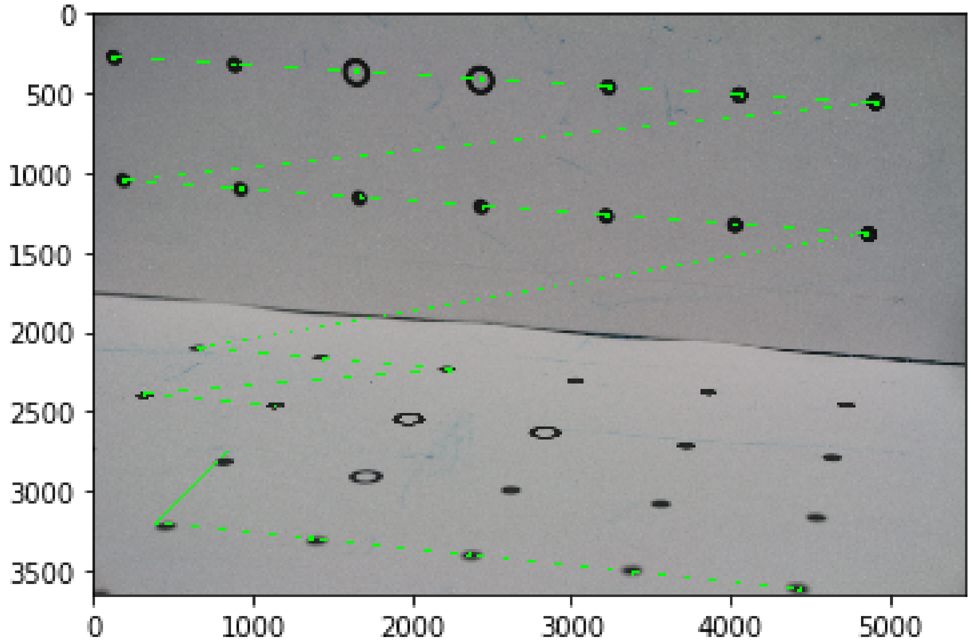
```
In [19]: M_ransac_undistorted = ransac_variation_calibration(u_img_points,u_world_points)
         print(M_ransac_undistorted)
```

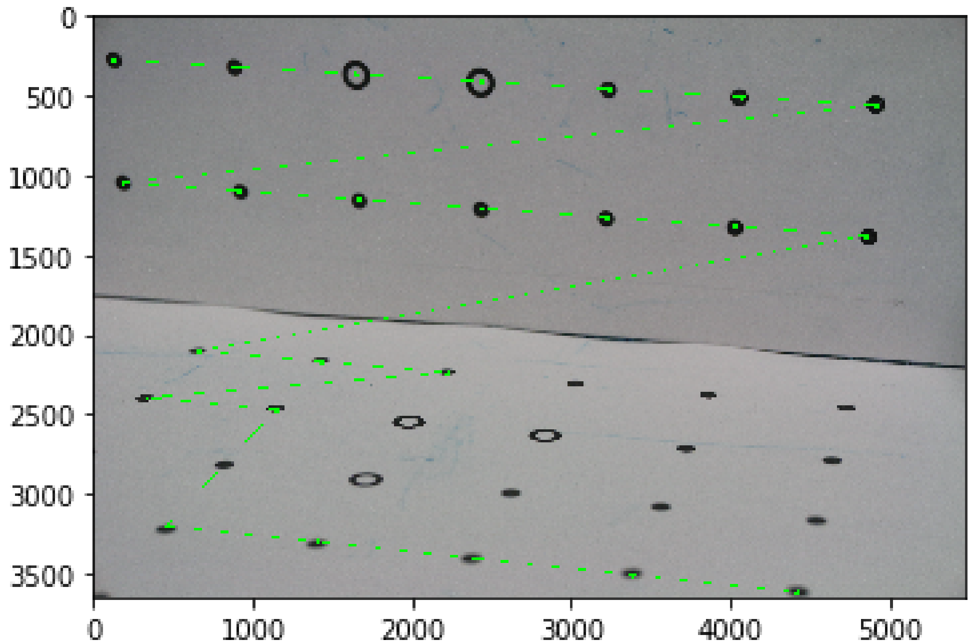
```
[[-4.06825958e-03 -2.34961360e-04 -1.86131350e-03  9.10830252e-01]
 [-2.14388480e-04 -4.34707227e-03  8.90461515e-04  4.12732856e-01]
 [ 7.24182934e-08 -9.63836526e-08 -2.89049322e-07  1.87540667e-04]]
```

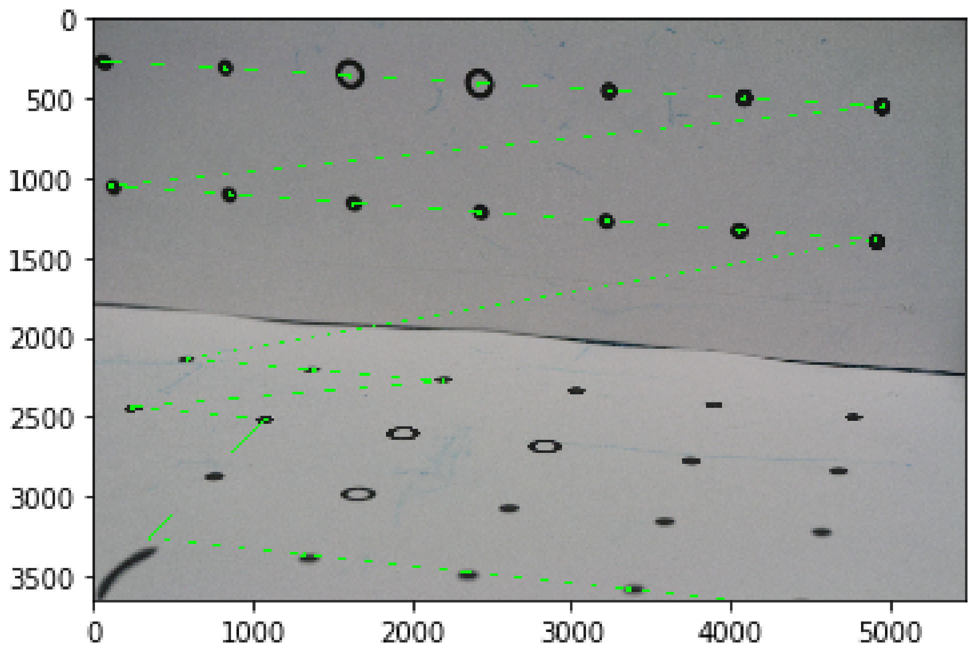
We can observe after correcting radial distortion and again performing calibration projection matrix got changed. It is not optimized for undistroted image.

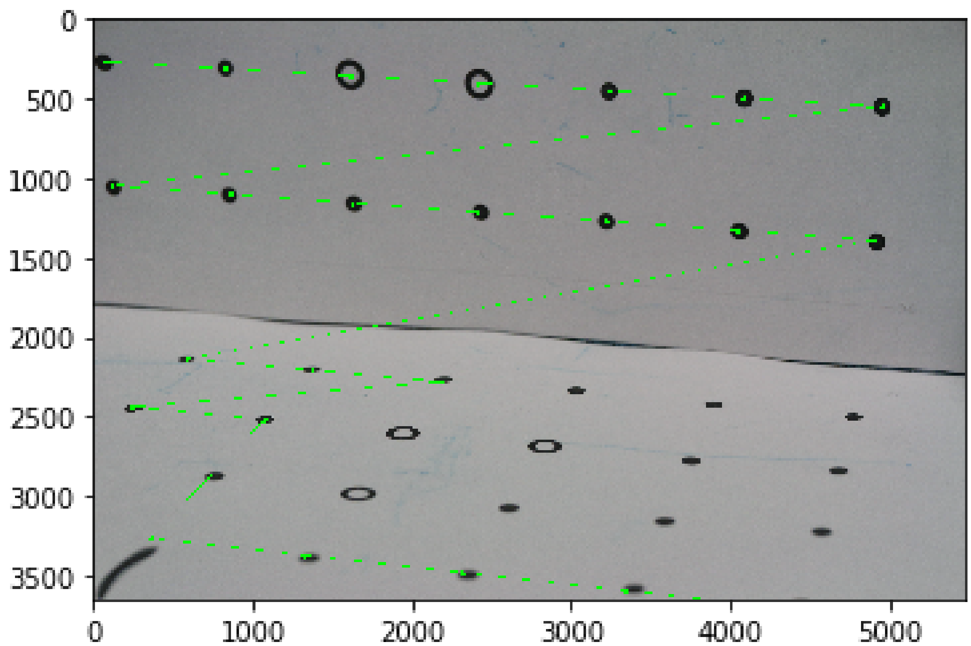
### 0.1.7 Answer 5 draw the wire frame

```
In [20]: gray = cv2.imread('../IMG_5455.JPG')
         draw_reproject_point(gray,world_points,M_DLT)
         draw_reproject_point(gray,world_points,M_ransac)
         gray = cv2.imread('../undistort.JPG')
         draw_reproject_point(gray,world_points,M_DLT_undistorted)
         draw_reproject_point(gray,world_points,M_ransac_undistorted)
```


0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000
5000


0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000
5000




overlay of undistored image will be streight line as compared to non distored image.

### 0.1.8 Answer 6 Zhangs method

```
In [21]: import glob

         board_w = 8
         board_h = 6        ## 7
         #Board dimensions (typically in cm)
         board_dim = 29

         path = '../images/*JPG'

In [22]: def detectcheckerborad(img_gray):
             thre_image = 255*((img_gray >50).astype('uint8'))
             return thre_image

         def inrinsicCalibration(path,flag):
             # termination criteria
             #criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.0001)
             criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.1)
             # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
         #    objp = np.zeros((9*7,3), np.float32)
         #    objp[:,:2] = np.mgrid[0:7,0:9].T.reshape(-1,2)




             objp = np.zeros((board_h*board_w,3), np.float32)
             objp[:,:2] = np.mgrid[0:(board_w*board_dim):board_dim,0:(board_h*board_dim):board_d

              # Arrays to store object points and image points from all the images.
             objpoints = [] # 3d point in real world space
             imgpoints = [] # 2d points in image plane.

             images = glob.glob(path)

             count = 1

             #textfile_W = open('Cam2_L'+ ".txt","w")
             fig=plt.figure(figsize=(16, 16))
             rows = 6
             columns = 2
             i=0
             for fname in sorted(images):

                 img = cv2.imread(fname)

                 gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
                 if(flag):
                     gray = detectcheckerborad(gray)
```

```
#        cv2.namedWindow('disparity', cv2.WINDOW_NORMAL)
#        cv2.resizeWindow('disparity', 800, 640)
#        cv2.imshow('disparity', gray)
#        cv2.waitKey()

        # Find the chess board corners
        ret, corners = cv2.findChessboardCorners(gray, (board_w,board_h),None)
        #print (ret)
        # If found, add object points, image points (after refining them)

        if ((ret == True) and (corners[(board_w*board_h)-1][0,1] > corners[0][0,1])):

            objpoints.append(objp)

            cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
            imgpoints.append(corners)
            #print (count)
                   # Draw and display the corners
            cv2.drawChessboardCorners(img, (board_w,board_h), corners,ret)
            #cv2.namedWindow('img', cv2.WINDOW_NORMAL)
            #cv2.resizeWindow('img', 800, 600)

            #cv2.imshow('img',img)
            #cv2.waitKey(100)
            #char = cv2.waitKey(0)

            fig.add_subplot(rows, columns, (i%5)+1)
            plt.axis("off")
            plt.title(str ((i%5)+1)+ " calibration image")
            plt.imshow(img)
            i=i+1
        count = count+1

    #        textfile_W.write(fname.split('\\')[1]+"\n")
    #textfile_W.close()
    cv2.destroyAllWindows()



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

    mean_error = 0

    tot_error =0
    total_points = 0
    error_list = []
```

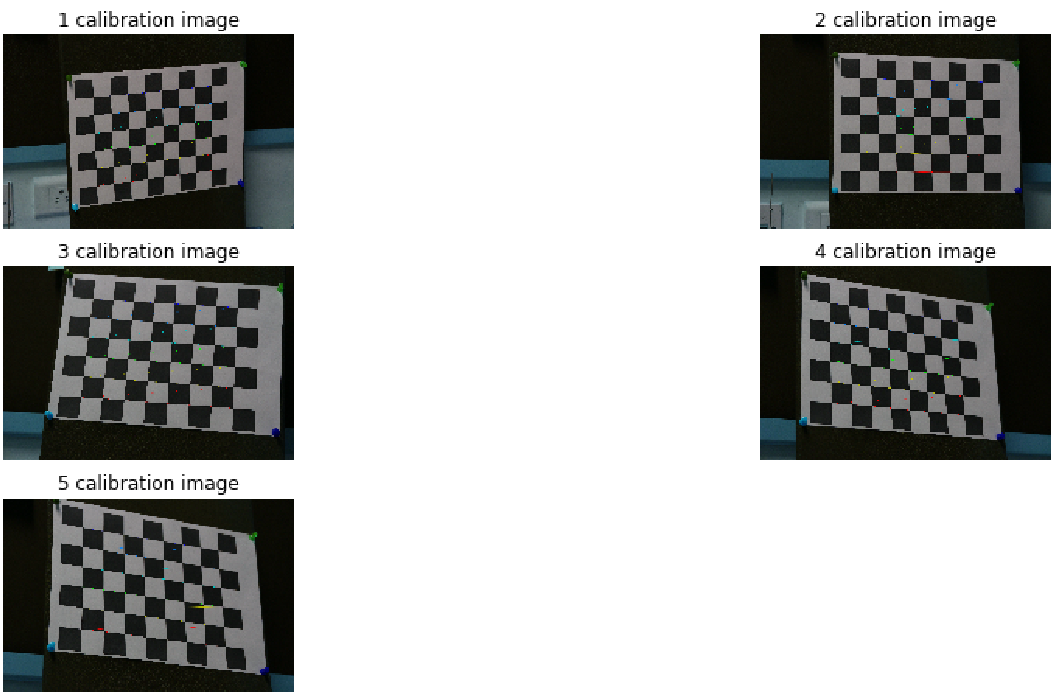
```
    for i in range(len(objpoints)):
        reprojected_points, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx
#        reprojected_points=reprojected_points.reshape(-1,2)
        tot_error+=np.sum(np.abs(imgpoints[i]-reprojected_points)**2)
        total_points+=len(objpoints[i])
        error_list.append(np.sqrt(np.sum(np.abs(imgpoints[i]-reprojected_points)**2)/le
    mean_error=np.sqrt(tot_error/total_points)
    print ("Mean reprojection error: ", mean_error)

    print(fname)
    return ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints, gray.shape[::-1],error_l
```

```
In [23]: ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints,gray_shape,error = inrinsicCalibrati
```

```
C:\Users\kandpani\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\cbook\deprecati
  warnings.warn(message, mplDeprecation, stacklevel=1)


Mean reprojection error:  2.333842789573212
../images\IMG_5470.JPG
```




### 0.1.9 6.2 how result campare

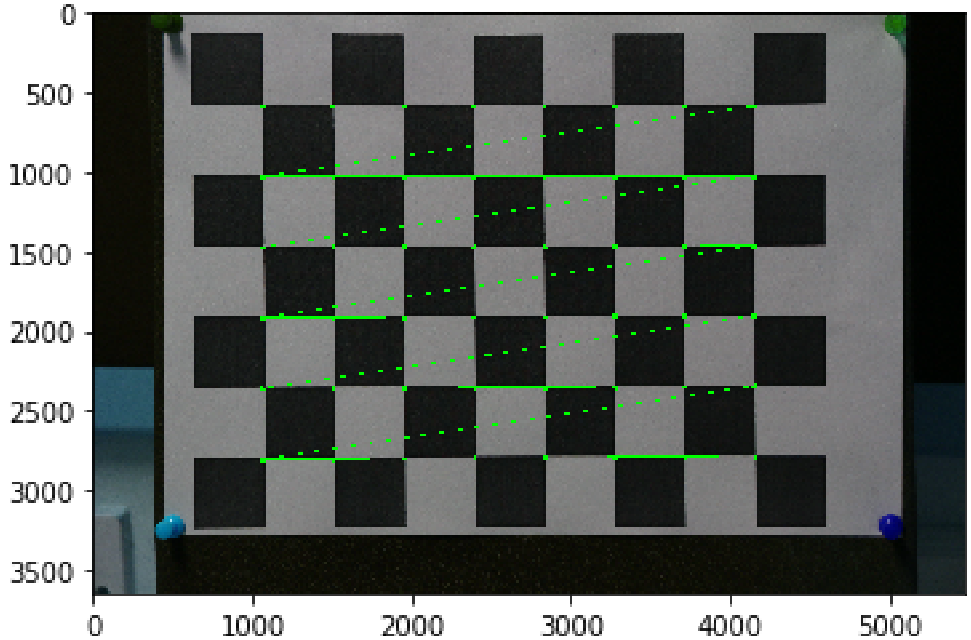
As compared to DLT method, Zhangs method gives the intrinsics in term of pixels value, so fx and fy all are in term of pixels. For many application it is better as we can do the calculation in pixels and even do the convesion from pixels to mm.

### 0.1.10 Answer 7 draw the points with projection matrix

```
In [24]: ### get the projection matrix for image 1 of checker board
         K = mtx
         R = cv2.Rodrigues(rvecs[0])[0]
         T = tvecs[0]
         RT = np.concatenate((R,T),axis=1)
         P = np.dot(K, RT)
```

```
In [25]: world_points = np.array(objpoints[0],dtype='float32')
```

```
In [26]: img = cv2.imread('../images/IMG_5456.JPG')
         draw_reproject_point(img,world_points,P)
```




### 0.1.11 Answer 8

Image of world origin mean camera at world origin so there will be no translation. T will be [0,0,0] in that case. IF we assume camera axis allined with world axis and camera at world origin than there will be no translation as weel as rotation. So Projection matrix will be P = K and RT will become the identity marix. In my observation i did not observe any such image as some translation or rotation was always present im my calibration images.

### 0.1.12 Answer 9

I am going to use gopro hero 5 camera.

### 0.1.13 Answer 10

Own camera calibration with zhange method

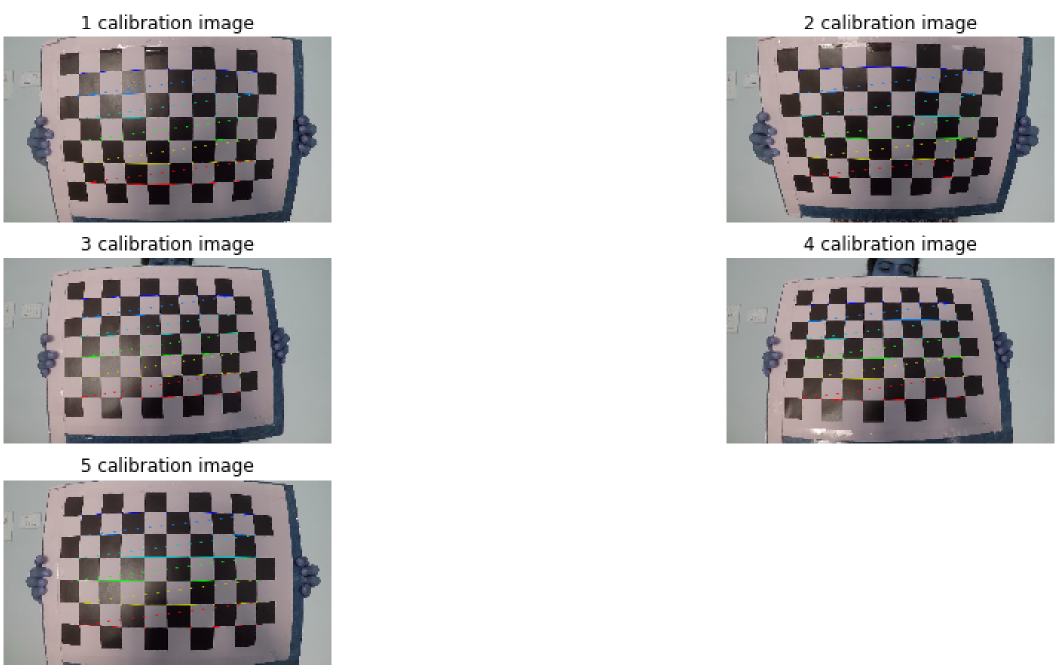
```
In [27]: board_w = 9
         board_h = 6        ## 7
         #Board dimensions (typically in cm)
         board_dim = 38

         path = '../own_images/*JPG'
```

```
In [28]: ret, mtx, dist, rvecs, tvecs, objpoints, imgpoints,gray_shape,error = inrinsicCalibrati
```

```
C:\Users\kandpani\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\cbook\deprecati
  warnings.warn(message, mplDeprecation, stacklevel=1)


Mean reprojection error:  0.7640167283485418
../own_images\847.JPG
```



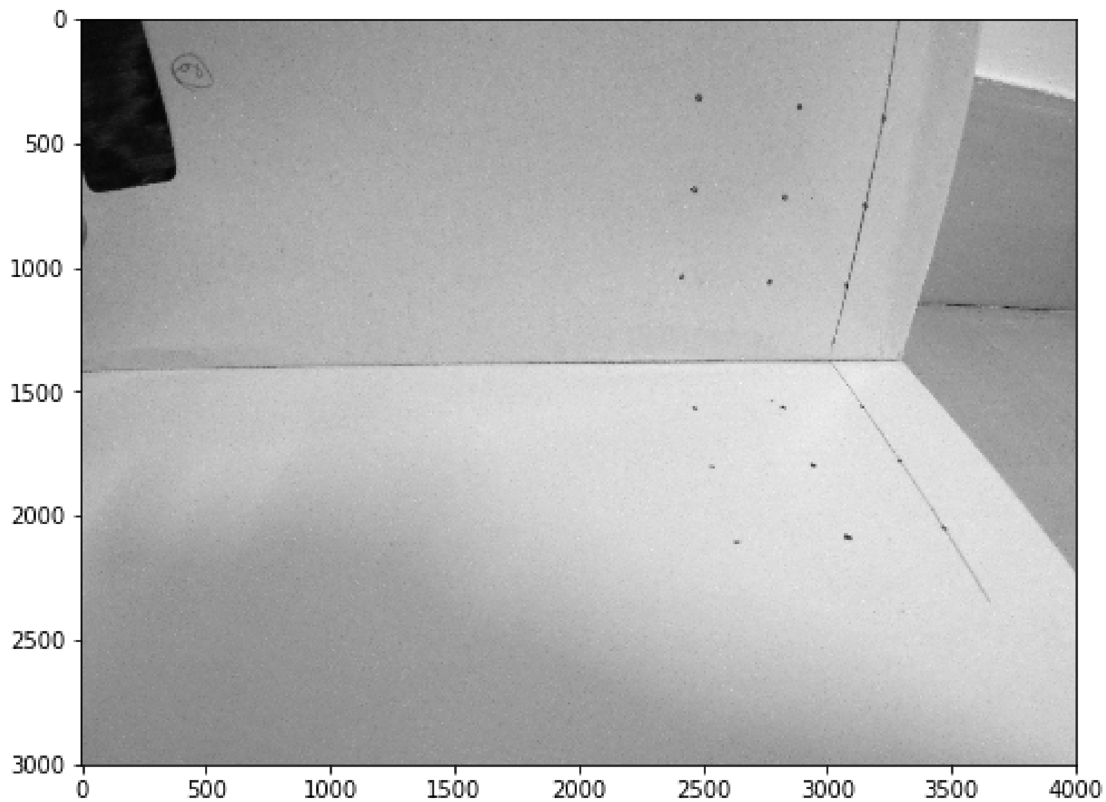

```
In [29]: mtx
```

```
Out[29]: array([[1.12759835e+03, 0.00000000e+00, 9.49656479e+02],
                [0.00000000e+00, 1.13584906e+03, 5.10121276e+02],
                [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

## 0.2 own_camera image for DLT and Ransac

In [30]:
```
img_DLT = cv2.imread('../DLT_on_image/DLT_image.JPG',0)
fig=plt.figure(figsize=(8, 8))
plt.imshow(img_DLT,cmap='gray')
```

Out[30]: <matplotlib.image.AxesImage at 0x2848ee1f470>




In [31]:
```
own_world_points = np.array([[50,75,0],[25,75,0],[0,75,0],[50,50,0],[25,50,0],[0,50,0],
                             [50,25,0],[25,25,0],[0,25,0],[50,0,25],[25,0,25],[0,0,25],
                             [50,0,50],[25,0,50],[0,0,50],[50,0,75],[25,0,75],[0,0,75]],d
own_image_points = np.array([[2481,317],[2889,349],[3225,397],[2465,685],[2892,717],[31
                              [2413,1037],[2769,1057],[3081,1073],[2469,1565],[2821,1561]
                               [2537,1801],[2949,1793],[3297,1777],[2637,2101],[3081,2081
```

In [32]:
```
M_DLT_own,_,_ = DLT_calibration(own_image_points,own_world_points)
print(M_DLT_own)
```

```
[[ 4.95343534e-03  9.15334624e-04  1.66082999e-03 -9.11529998e-01]
 [ 6.77510955e-04  4.08846954e-03 -7.06308059e-04 -4.11177780e-01]
 [ 4.61235399e-07  5.32281201e-07  9.82053345e-07 -3.00564708e-04]]
```

```
In [33]: M_ransac_own = ransac_variation_calibration(own_image_points,own_world_points)
         print(M_ransac_own)
```

```
[[ 5.09806962e-03  9.59001100e-04  1.67956955e-03 -9.12827047e-01]
 [ 7.04876745e-04  4.04003103e-03 -7.28472103e-04 -4.08288636e-01]
 [ 5.10561234e-07  5.47298517e-07  9.87155253e-07 -3.00740369e-04]]
```

from zhange method we can say my camera focal length fx is 1127 and fy is 1135 in term of pixels. There are some diffrence in projection matrix of DLT method abd ransac variation method values. This is due to poor pixel selection. There are direction diffrence in projection matrix due to effect of rotation matrix.