Problem 1 : Implement the DLT (Direct Linear Transformation) based calibration that we discussed in the class.

This algorithm takes N points with World and Image correspondences and then solve the equation for finding Projection Method as follows:

Algorithm

$$\mathbf{x} = P\mathbf{X}$$

(P is Projection Matrix, **X** is Homogeneous World Points, **x** is Homogeneous Image Points**)** is rearranged such that it changes to AP1 = 0 where A is computed from world points and Image Points. P1 is the flattened P matrix. A is of the form 2n X 12 where n is number of image points.

Next SVD of A gives U,D,V^T where the last column of V gives the projection matrix so that least eigenvalue is picked. Reshaping this to 3X4 form gives us the projection matrix. It is then Normalized to make P[3,4] = 1

Further QR decomposition of obtained P matrix gives us the Camera Matrix (K) and Rotation Matrix(R). This is then used to find the Camera Projection Matrix(T).
Here are the observed results on the given images.

Projection Matrix is given as :

```
[[-2.11225340e+01 -1.52511970e+00 -9.72155998e+00  4.82458631e+03]
 [-1.09832879e+00 -2.24096507e+01  4.43993421e+00  2.15758505e+03]
 [ 3.60153741e-04 -6.15323259e-04 -1.57382436e-03  1.00000000e+00]]
```

Camera Matrix is:

```
[[-1.31727418e+04  1.10459423e+02  2.89124562e+03]
 [-0.00000000e+00 -1.30624295e+04  2.14585760e+03]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

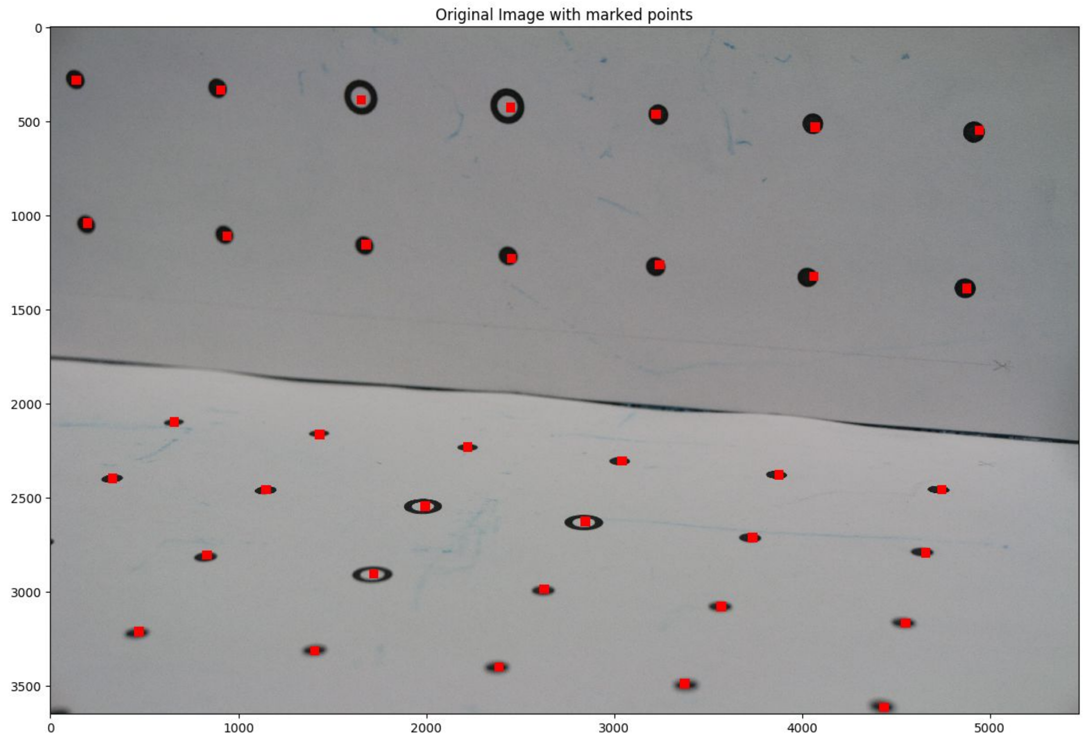
Rotation Matrix is :

```
[[-0.9745131   0.00332142 -0.22430602]
 [-0.08290827 -0.93442929  0.34636416]
 [-0.2084477   0.35613323  0.91088895]]
```

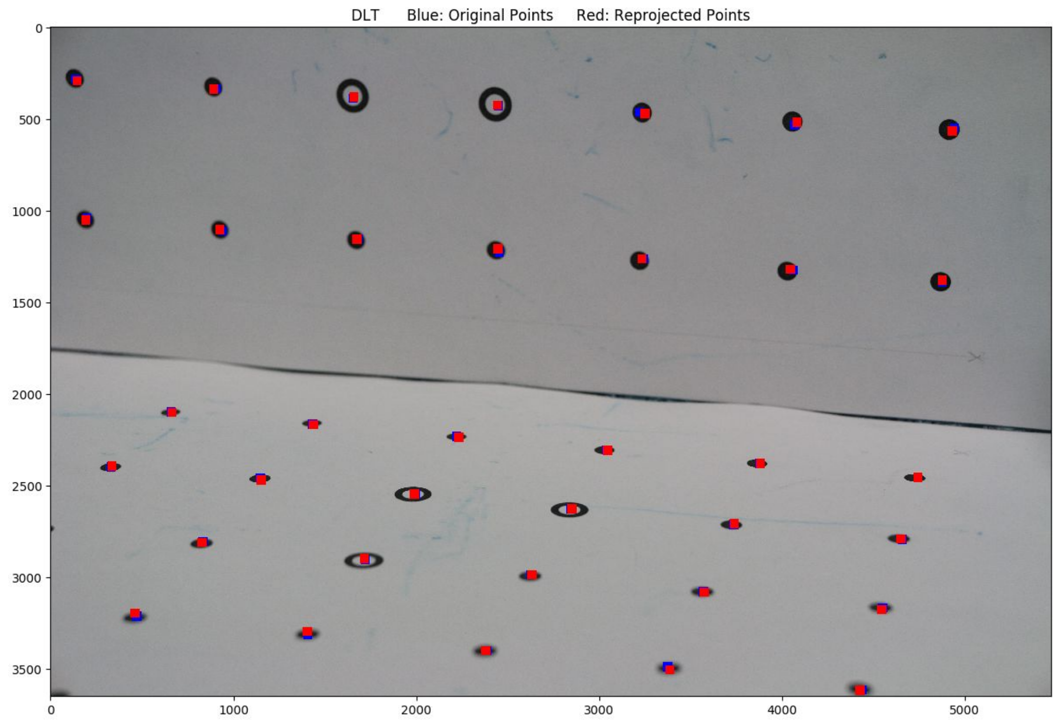
Projection Center is:

```
[-37.81612898 206.32412608 546.07386062]
```

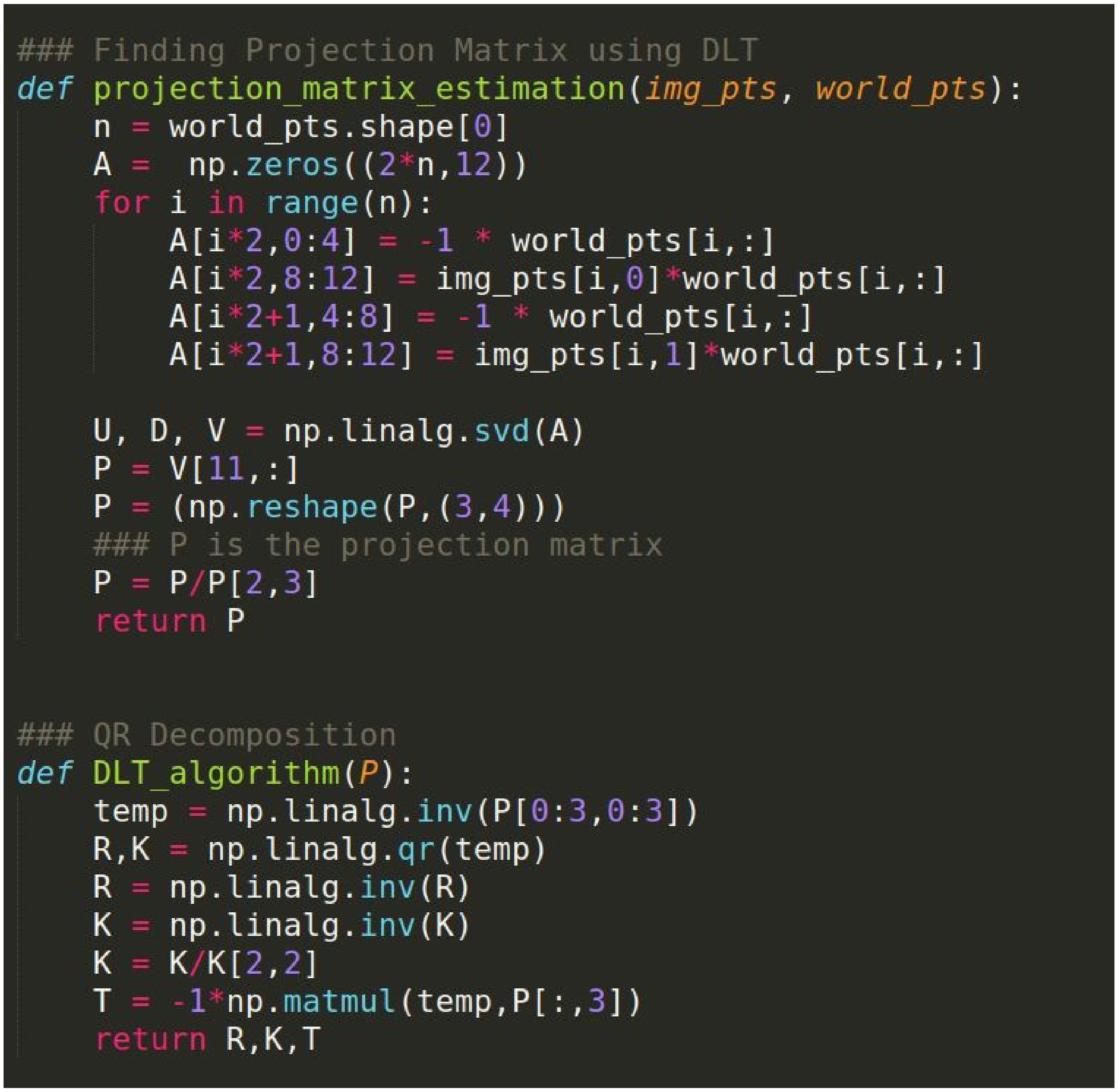
Original Points manually taken:



Observed Reprojections and Image points from P obtained by DLT Method:



Code Snippet:



```
### Finding Projection Matrix using DLT
def projection_matrix_estimation(img_pts, world_pts):
    n = world_pts.shape[0]
    A =  np.zeros((2*n,12))
    for i in range(n):
        A[i*2,0:4] = -1 * world_pts[i,:]
        A[i*2,8:12] = img_pts[i,0]*world_pts[i,:]
        A[i*2+1,4:8] = -1 * world_pts[i,:]
        A[i*2+1,8:12] = img_pts[i,1]*world_pts[i,:]

    U, D, V = np.linalg.svd(A)
    P = V[11,:]
    P = (np.reshape(P,(3,4)))
    ### P is the projection matrix
    P = P/P[2,3]
    return P


### QR Decomposition
def DLT_algorithm(P):
    temp = np.linalg.inv(P[0:3,0:3])
    R,K = np.linalg.qr(temp)
    R = np.linalg.inv(R)
    K = np.linalg.inv(K)
    K = K/K[2,2]
    T = -1*np.matmul(temp,P[:,3])
    return R,K,T
```

Problem 2 , 3 :Implement the RANSAC based variant of the calibration that we discussed in the class. Note that these two algorithms use a set of known correspondences between real-world points and image points.

RANSAC (RANdom SAmpling Consensus) is a resampling technique which is used to find the best possible minimum candidates required for finding P matrix by using only 6 points to find P matrix and then finding number of points which are under a threshold for the reprojected points and image points.

This process is repeated for N = 2000 number of times and then the best P matrix is used for finding R and T.

Here are the observations:

Projection Matrix after RANSAC is:

```
[[-2.11333929e+01 -6.32870256e-01 -9.57166082e+00  4.81544132e+03]
 [-1.01650851e+00 -2.18160281e+01  4.55238697e+00  2.14441851e+03]
 [ 3.65175587e-04 -4.86556493e-04 -1.56445244e-03  1.00000000e+00]]
```

Camera Matrix is :

```
[[-1.35558372e+04  4.46601308e+02  2.68488468e+03]
 [-0.00000000e+00 -1.32442816e+04  1.10786878e+03]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

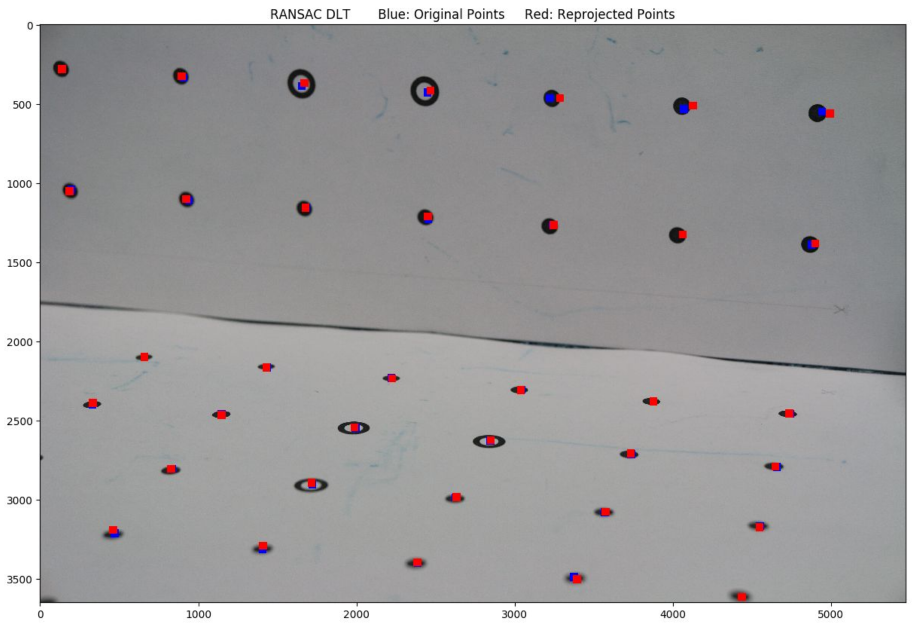
Rotation Matrix is :

```
[[-0.97395348 -0.0019332  -0.22673967]
 [-0.06392177 -0.95706606  0.28273409]
 [-0.21755142  0.28986345  0.93201425]]
```

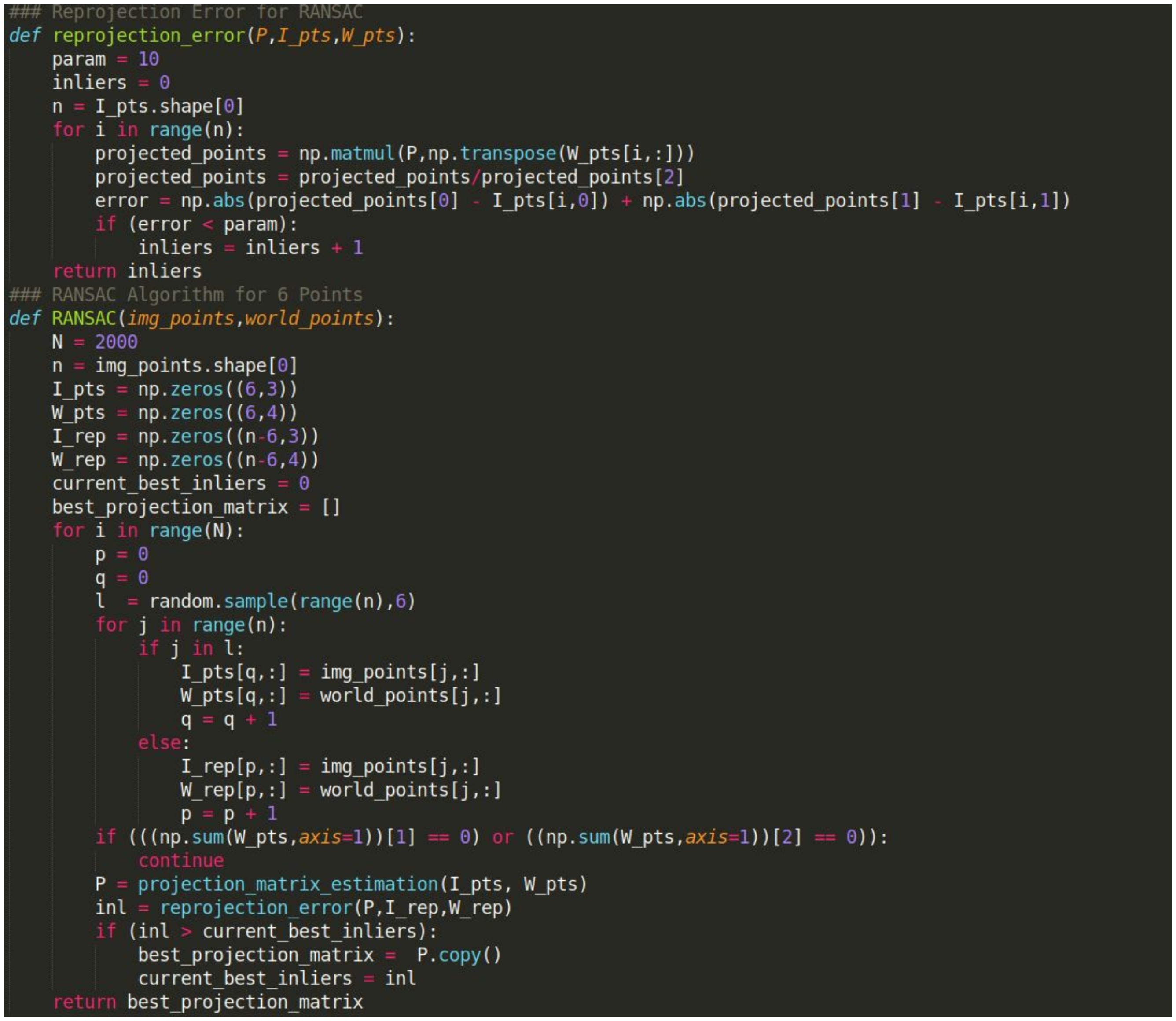
Projection Center is:

```
[-33.93489057 217.49211527 563.63848551]
```

The P matrix was then used to reproject the points and they are plotted on image as shown. Blue marker represents the Original Points and Red represents the Reprojected Points.



Code Snippet:

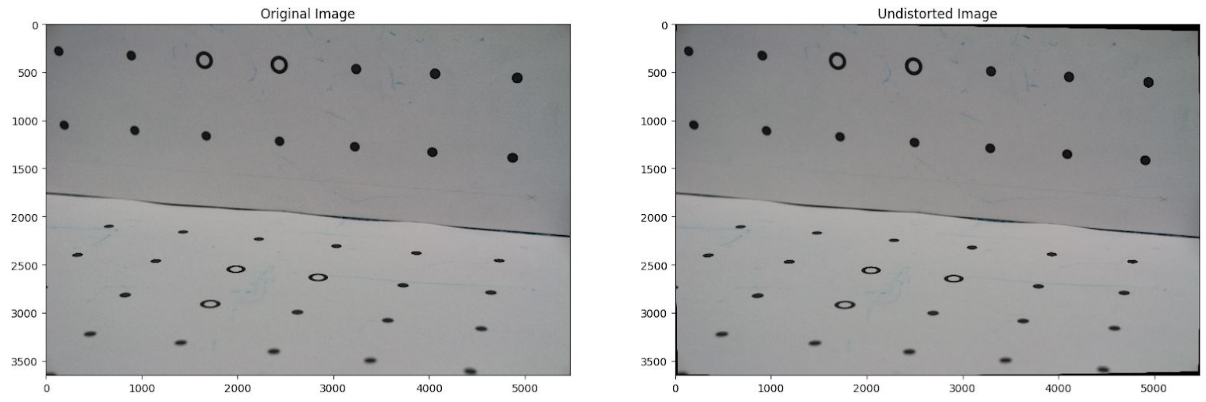


```
### Reprojection Error for RANSAC
def reprojection_error(P,I_pts,W_pts):
    param = 10
    inliers = 0
    n = I_pts.shape[0]
    for i in range(n):
        projected_points = np.matmul(P,np.transpose(W_pts[i,:]))
        projected_points = projected_points/projected_points[2]
        error = np.abs(projected_points[0] - I_pts[i,0]) + np.abs(projected_points[1] - I_pts[i,1])
        if (error < param):
            inliers = inliers + 1
    return inliers
### RANSAC Algorithm for 6 Points
def RANSAC(img_points,world_points):
    N = 2000
    n = img_points.shape[0]
    I_pts = np.zeros((6,3))
    W_pts = np.zeros((6,4))
    I_rep = np.zeros((n-6,3))
    W_rep = np.zeros((n-6,4))
    current_best_inliers = 0
    best_projection_matrix = []
    for i in range(N):
        p = 0
        q = 0
        l  = random.sample(range(n),6)
        for j in range(n):
            if j in l:
                I_pts[q,:] = img_points[j,:]
                W_pts[q,:] = world_points[j,:]
                q = q + 1
            else:
                I_rep[p,:] = img_points[j,:]
                W_rep[p,:] = world_points[j,:]
                p = p + 1
        if (((np.sum(W_pts,axis=1))[1] == 0) or ((np.sum(W_pts,axis=1))[2] == 0)):
            continue
        P = projection_matrix_estimation(I_pts, W_pts)
        inl = reprojection_error(P,I_rep,W_rep)
        if (inl > current_best_inliers):
            best_projection_matrix =  P.copy()
            current_best_inliers = inl
    return best_projection_matrix
```

Problem 4:
Radial Distortion is corrected and the plot shows the original and distorted image.

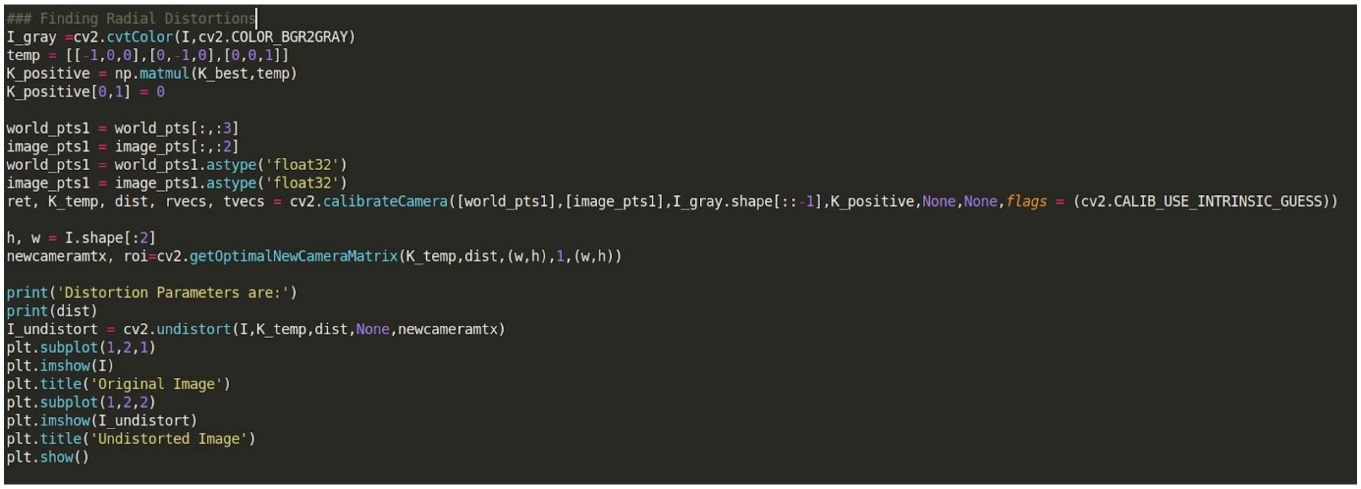
Distortion Parameters are:

```
[[ 6.90747433e-01 -7.92332549e+00  1.20551292e-02  6.50107752e-02
   4.30052966e+01]]
```



Since the distortions observed are very small, hence the observed results after distortions are very close to observed results.

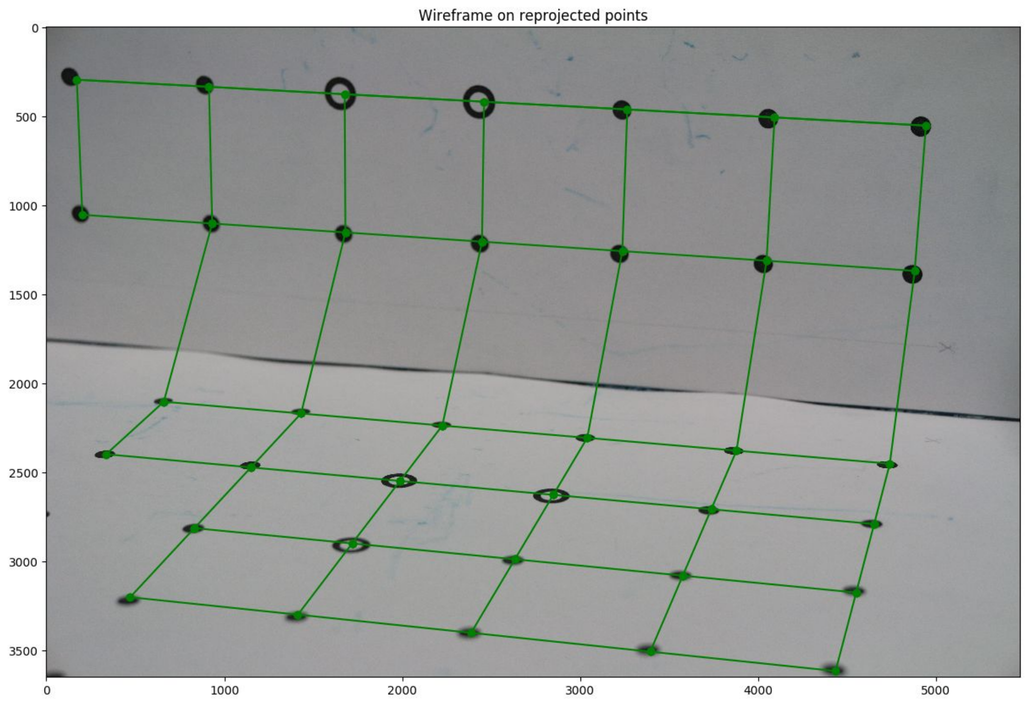
Code Snippet:



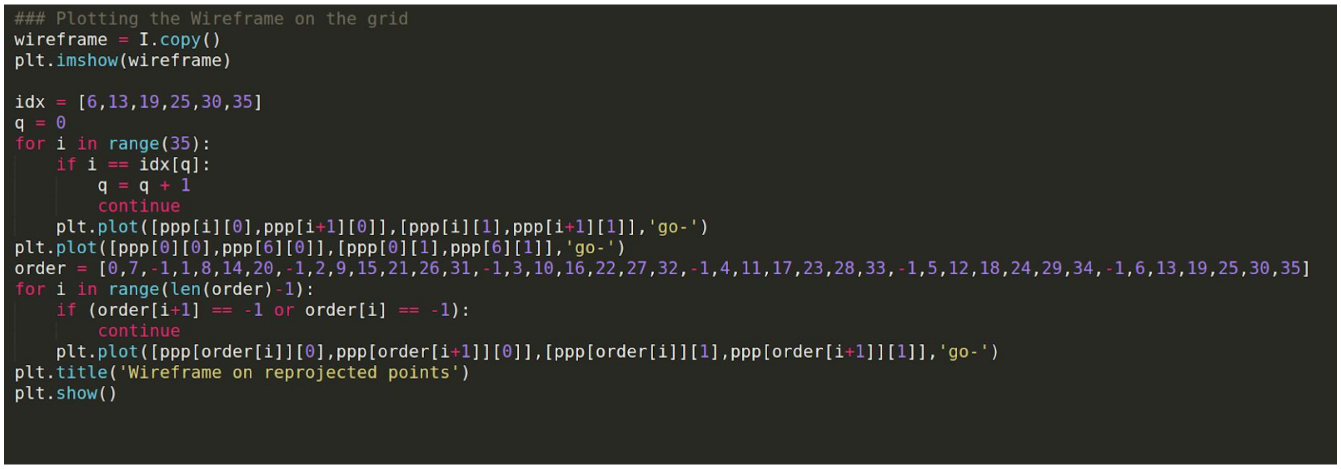
```
### Finding Radial Distortions
I_gray =cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
temp = [[-1,0,0],[0,-1,0],[0,0,1]]
K_positive = np.matmul(K_best,temp)
K_positive[0,1] = 0

world_pts1 = world_pts[:,:3]
image_pts1 = image_pts[:,:2]
world_pts1 = world_pts1.astype('float32')
image_pts1 = image_pts1.astype('float32')
ret, K_temp, dist, rvecs, tvecs = cv2.calibrateCamera([world_pts1],[image_pts1],I_gray.shape[::-1],K_positive,None,None,flags = (cv2.CALIB_USE_INTRINSIC_GUESS))

h, w = I.shape[:2]
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(K_temp,dist,(w,h),1,(w,h))

print('Distortion Parameters are:')
print(dist)
I_undistort = cv2.undistort(I,K_temp,dist,None,newcameramtx)
plt.subplot(1,2,1)
plt.imshow(I)
plt.title('Original Image')
plt.subplot(1,2,2)
plt.imshow(I_undistort)
plt.title('Undistorted Image')
plt.show()
```

Problem 5: Use the real-world measurements that are provided along with the estimated camera parameters to compute the image of a wireframe of the object. Note that you will be computing the location of image points as xi = P:Xi, and not use the image points. Overlay (draw) the wireframe over the actual image of the object using straight lines between the computed points xi. What do you observe about the overlay?

The P matrix is then used to reproject the image points and then a wireframe is computed and plotted on the image.



Code Snippet:



```
### Plotting the Wireframe on the grid
wireframe = I.copy()
plt.imshow(wireframe)

idx = [6,13,19,25,30,35]
q = 0
for i in range(35):
    if i == idx[q]:
        q = q + 1
        continue
    plt.plot([ppp[i][0],ppp[i+1][0]],[ppp[i][1],ppp[i+1][1]],'go-')
plt.plot([ppp[0][0],ppp[6][0]],[ppp[0][1],ppp[6][1]],'go-')
order = [0,7,-1,1,8,14,20,-1,2,9,15,21,26,31,-1,3,10,16,22,27,32,-1,4,11,17,23,28,33,-1,5,12,18,24,29,34,-1,6,13,19,25,30,35]
for i in range(len(order)-1):
    if (order[i+1] == -1 or order[i] == -1):
        continue
    plt.plot([ppp[order[i]][0],ppp[order[i+1]][0]],[ppp[order[i]][1],ppp[order[i+1]][1]],'go-')
plt.title('Wireframe on reprojected points')
plt.show()
```

Problem 6: Repeat the calibration of the camera using Zhang's method using either the available OpenCV or Matlab implementation. How do your results compare with those of the DLT based method?

For the implementation of Zhang's Method a 8X6 checkerboard is used and an inbuilt OpenCV implementation is used.
Observations obtained are:
Camera Calibration Matrix:

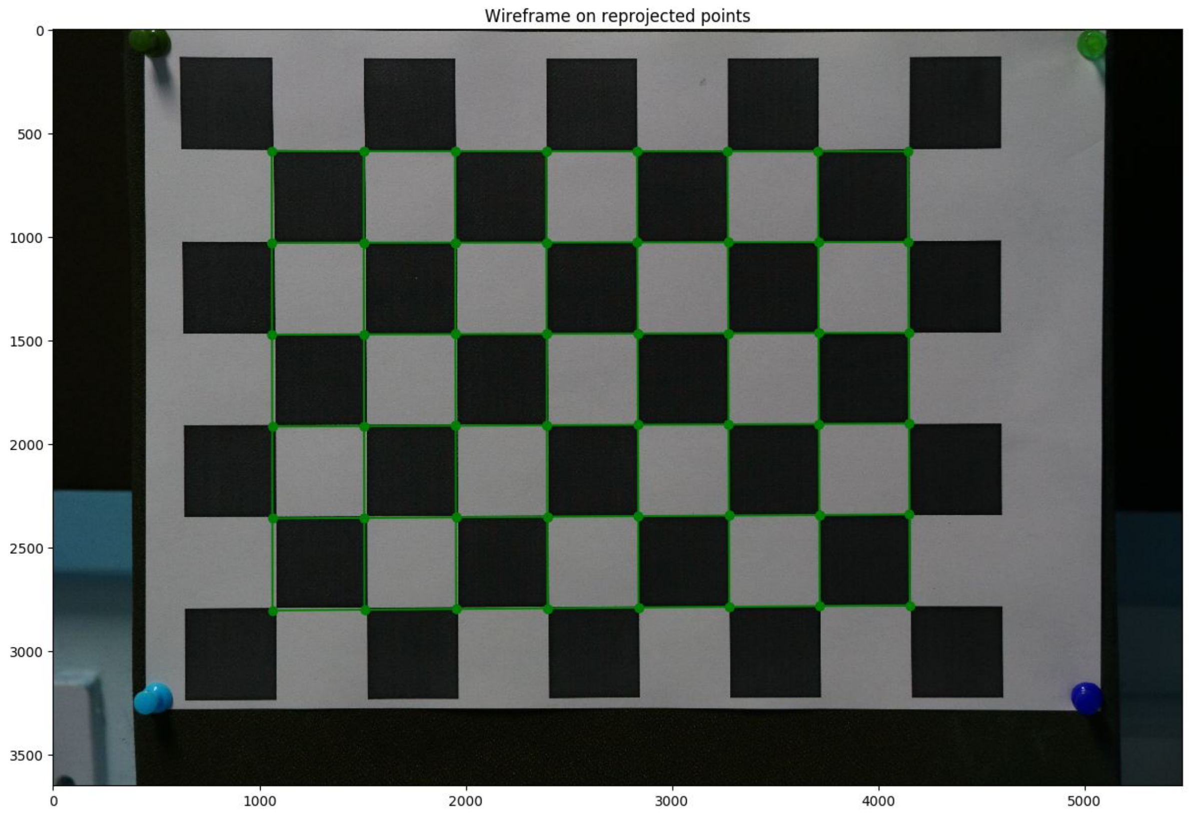
```
[[1.36634816e+04 0.00000000e+00 3.33651275e+03]
 [0.00000000e+00 1.36813888e+04 1.49657985e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

Distortion Parameters:

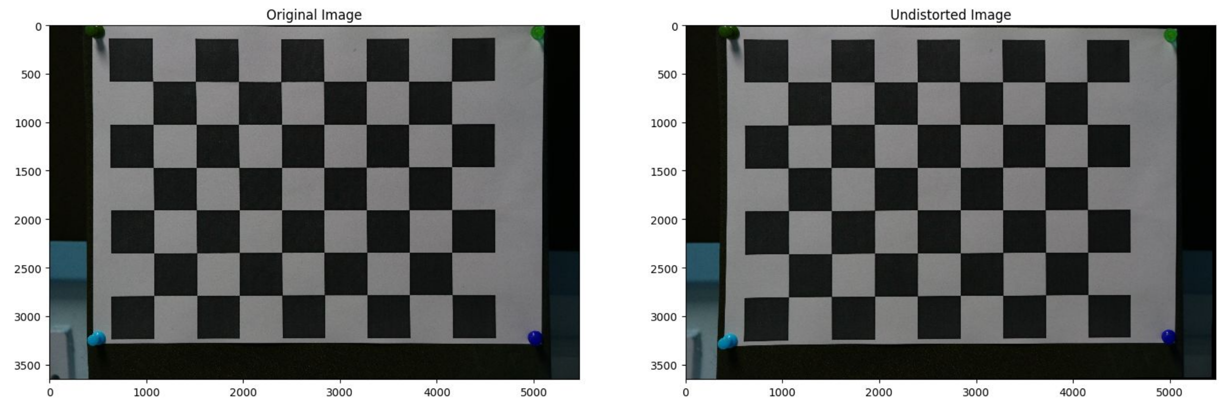
```
[[ 9.51409607e-02  1.01150408e+01 -1.52797290e-02  2.87204086e-02
  -1.60701382e+02]]
```

So as we can see that the focal length along x axis and y axis are almost similar in both the methods. However there is a variation in the camera principal points and scaling.

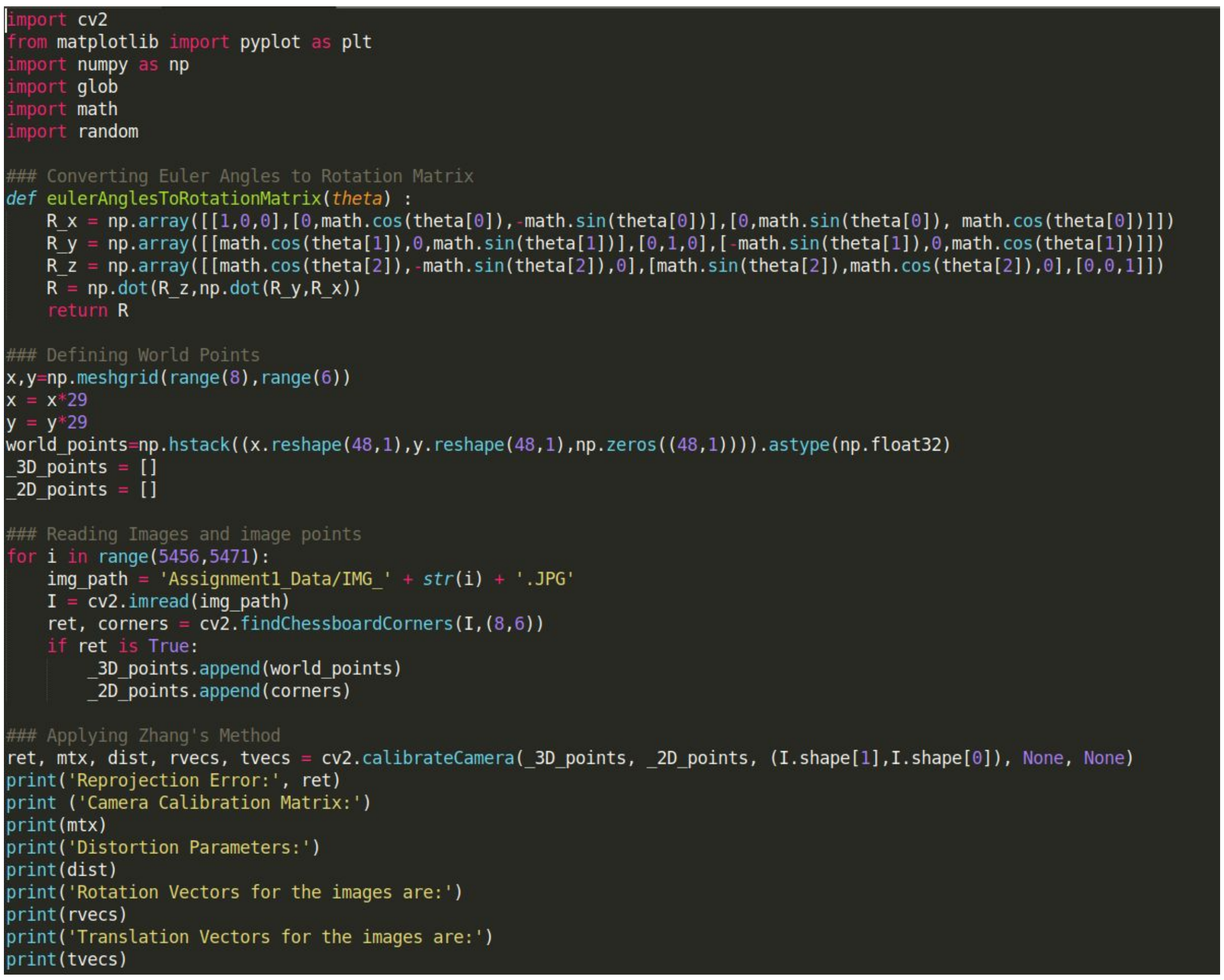
Problem 7: Repeat the overlay of wireframe using the internal parameters estimated from Zhang's method. Describe your observations.



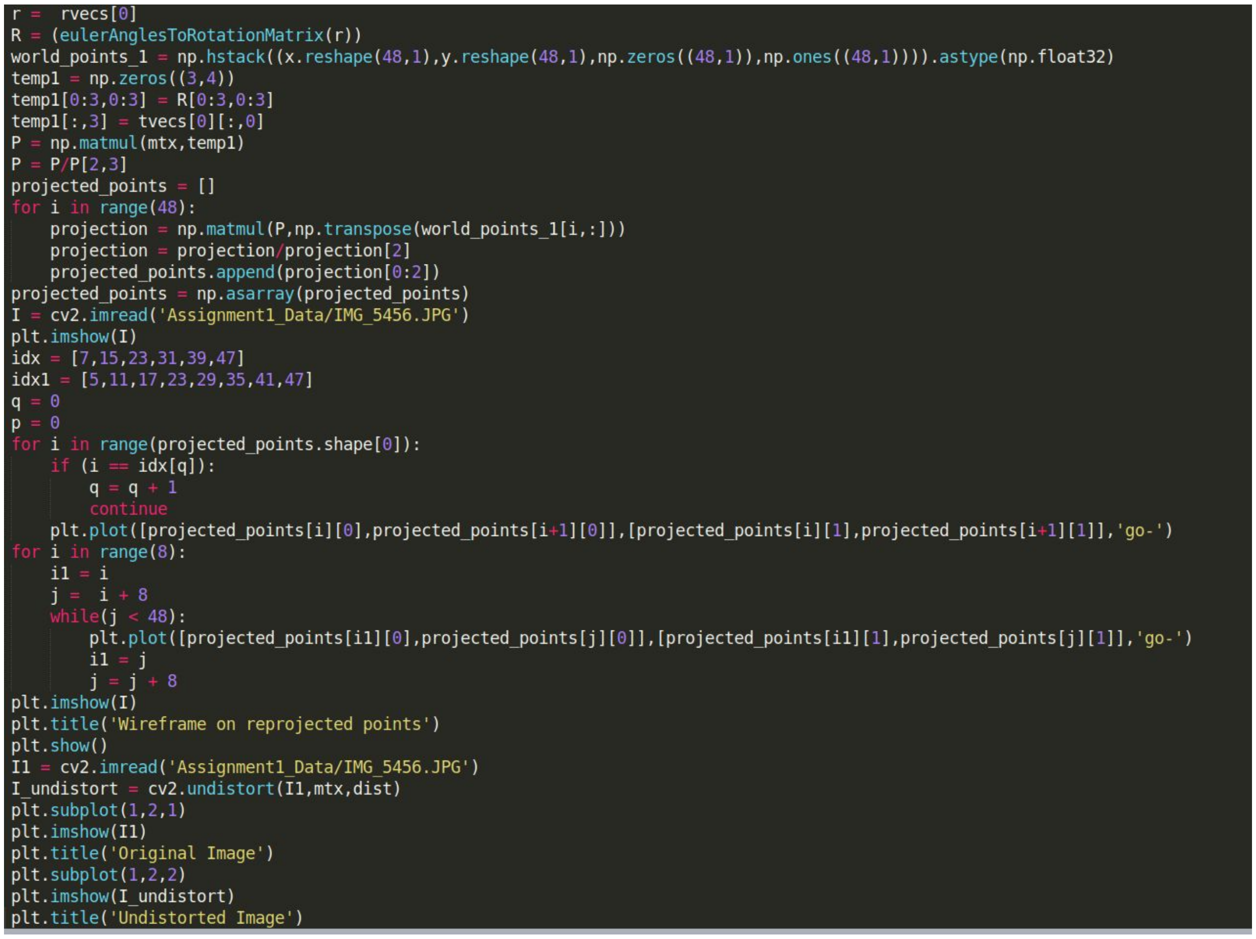
Original and Undistorted Image



Code Snippet:



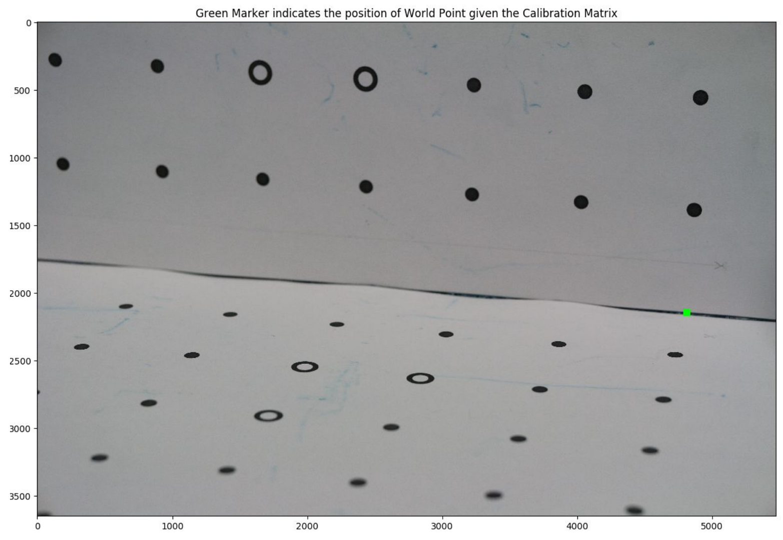
```python
import cv2
from matplotlib import pyplot as plt
import numpy as np
import glob
import math
import random

### Converting Euler Angles to Rotation Matrix
def eulerAnglesToRotationMatrix(theta) :
    R_x = np.array([[1,0,0],[0,math.cos(theta[0]),-math.sin(theta[0])],[0,math.sin(theta[0]), math.cos(theta[0])]])
    R_y = np.array([[math.cos(theta[1]),0,math.sin(theta[1])],[0,1,0],[-math.sin(theta[1]),0,math.cos(theta[1])]])
    R_z = np.array([[math.cos(theta[2]),-math.sin(theta[2]),0],[math.sin(theta[2]),math.cos(theta[2]),0],[0,0,1]])
    R = np.dot(R_z,np.dot(R_y,R_x))
    return R

### Defining World Points
x,y=np.meshgrid(range(8),range(6))
x = x*29
y = y*29
world_points=np.hstack((x.reshape(48,1),y.reshape(48,1),np.zeros((48,1)))).astype(np.float32)
_3D_points = []
_2D_points = []

### Reading Images and image points
for i in range(5456,5471):
    img_path = 'Assignment1_Data/IMG_' + str(i) + '.JPG'
    I = cv2.imread(img_path)
    ret, corners = cv2.findChessboardCorners(I,(8,6))
    if ret is True:
        _3D_points.append(world_points)
        _2D_points.append(corners)

### Applying Zhang's Method
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(_3D_points, _2D_points, (I.shape[1],I.shape[0]), None, None)
print('Reprojection Error:', ret)
print ('Camera Calibration Matrix:')
print(mtx)
print('Distortion Parameters:')
print(dist)
print('Rotation Vectors for the images are:')
print(rvecs)
print('Translation Vectors for the images are:')
print(tvecs)
```



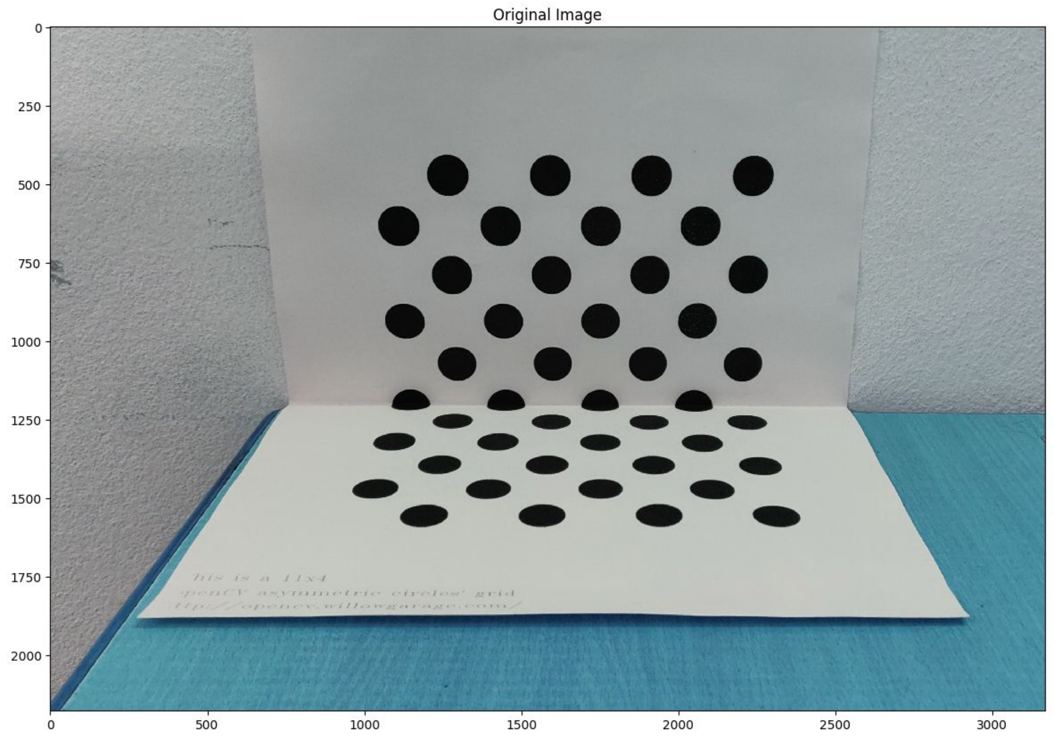
```
r =  rvecs[0]
R = (eulerAnglesToRotationMatrix(r))
world_points_1 = np.hstack((x.reshape(48,1),y.reshape(48,1),np.zeros((48,1)),np.ones((48,1)))).astype(np.float32)
temp1 = np.zeros((3,4))
temp1[0:3,0:3] = R[0:3,0:3]
temp1[:,3] = tvecs[0][:,0]
P = np.matmul(mtx,temp1)
P = P/P[2,3]
projected_points = []
for i in range(48):
    projection = np.matmul(P,np.transpose(world_points_1[i,:]))
    projection = projection/projection[2]
    projected_points.append(projection[0:2])
projected_points = np.asarray(projected_points)
I = cv2.imread('Assignment1_Data/IMG_5456.JPG')
plt.imshow(I)
idx = [7,15,23,31,39,47]
idx1 = [5,11,17,23,29,35,41,47]
q = 0
p = 0
for i in range(projected_points.shape[0]):
    if (i == idx[q]):
        q = q + 1
        continue
    plt.plot([projected_points[i][0],projected_points[i+1][0]],[projected_points[i][1],projected_points[i+1][1]],'go-')
for i in range(8):
    i1 = i
    j =  i + 8
    while(j < 48):
        plt.plot([projected_points[i1][0],projected_points[j][0]],[projected_points[i1][1],projected_points[j][1]],'go-')
        i1 = j
        j = j + 8
plt.imshow(I)
plt.title('Wireframe on reprojected points')
plt.show()
I1 = cv2.imread('Assignment1_Data/IMG_5456.JPG')
I_undistort = cv2.undistort(I1,mtx,dist)
plt.subplot(1,2,1)
plt.imshow(I1)
plt.title('Original Image')
plt.subplot(1,2,2)
plt.imshow(I_undistort)
plt.title('Undistorted Image')
```

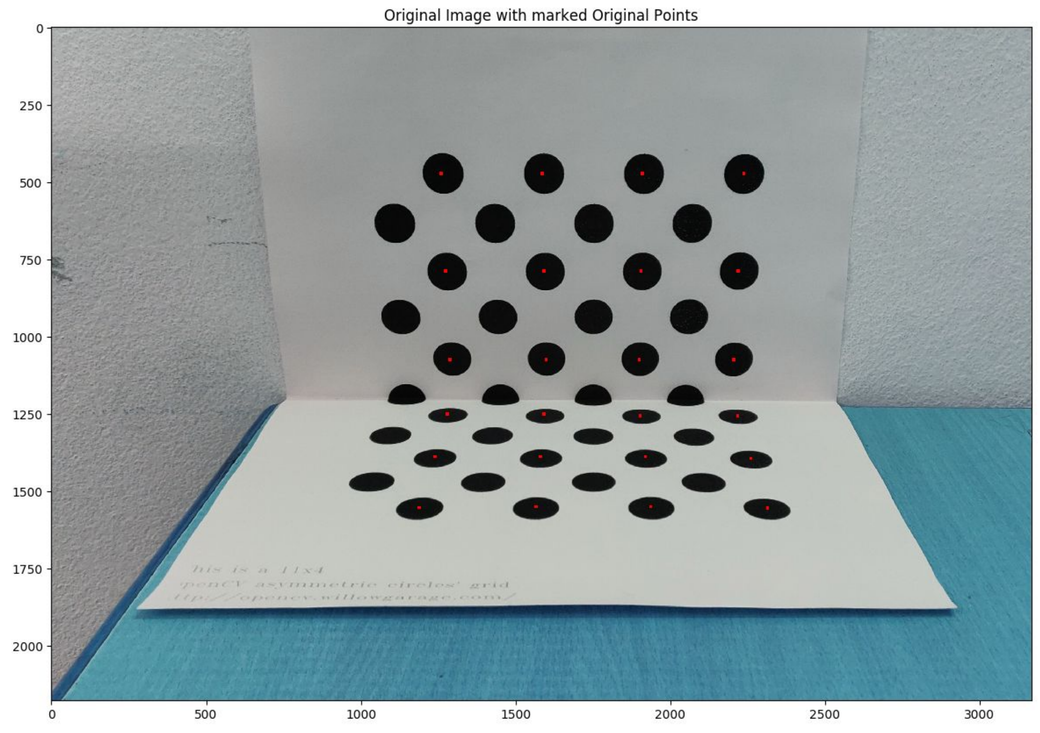
Problem 8: What is the image of the world origin, given the calibration matrix? Does this result bear out in your observations?

Image of the world origin should be the projection centre of the image.



Problem 9,10 : Repeating the above experiments with our own camera.




Original Image with marked Original Points
0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500
3000
his is a 11x4
penCV asymmetric circles' grid
ttp://opencv.willowgarage.com/

DLT Method

Projection Matrix is given as :

```
[[-5.93398500e+01 -1.35790526e+01 -2.93550697e+01  2.19391978e+03]
 [-8.91384989e-02 -6.18576811e+01  1.52515466e+00  1.20434345e+03]
 [ 4.40015462e-05 -7.80115741e-03 -1.69275534e-02  1.00000000e+00]]
```

Camera Matrix is :

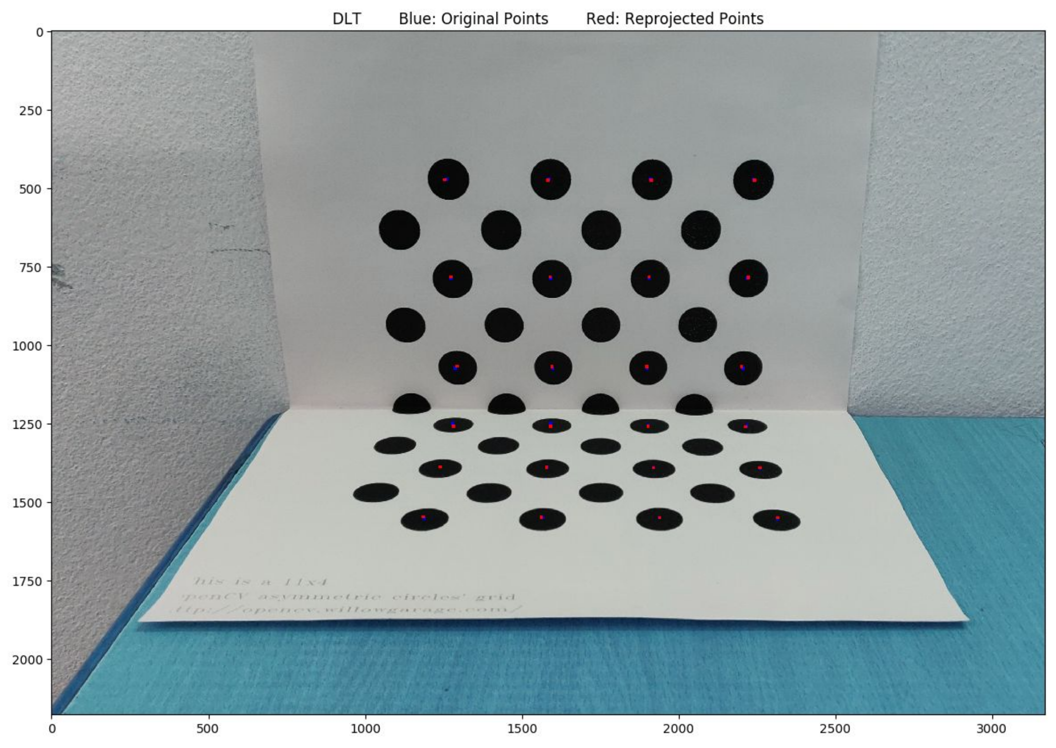
```
[[-3.18775686e+03 -1.07126962e+01  1.72776982e+03]
 [-0.00000000e+00 -3.04835174e+03  1.31473155e+03]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

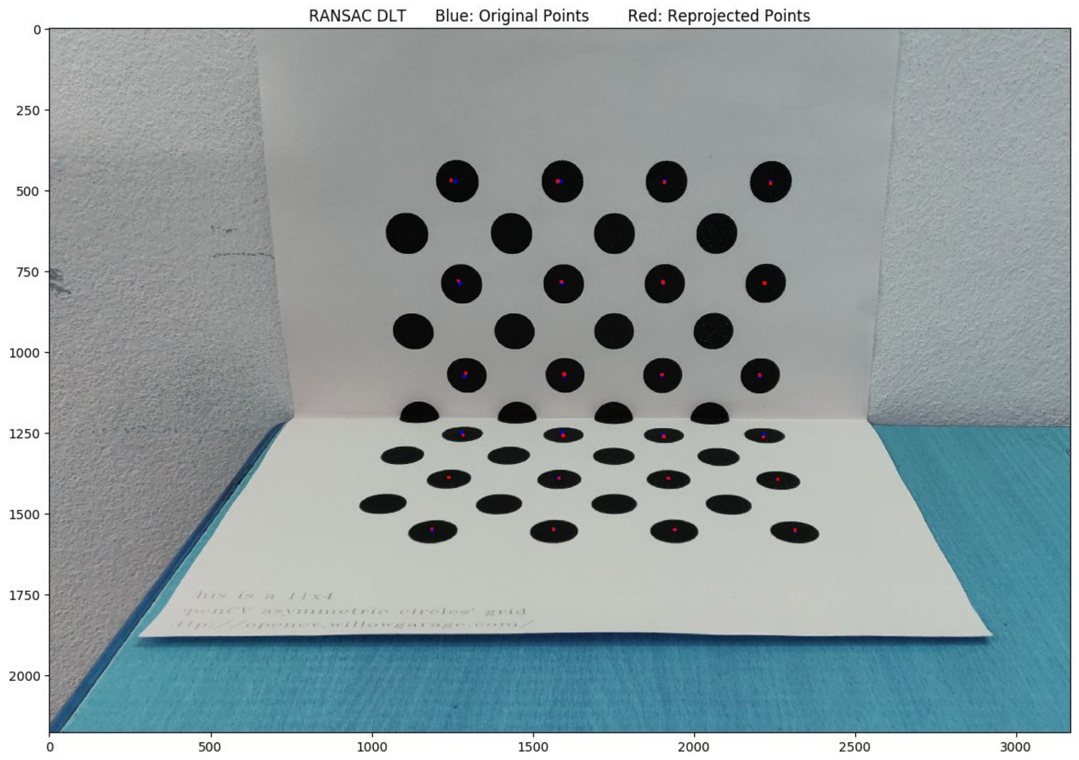
Rotation Matrix is :

```
[[-0.99999387  0.00136146 -0.00322682]
 [-0.00258704 -0.90819473  0.41853988]
 [-0.00236076  0.41854566  0.90819269]]
```

Projection Center is:

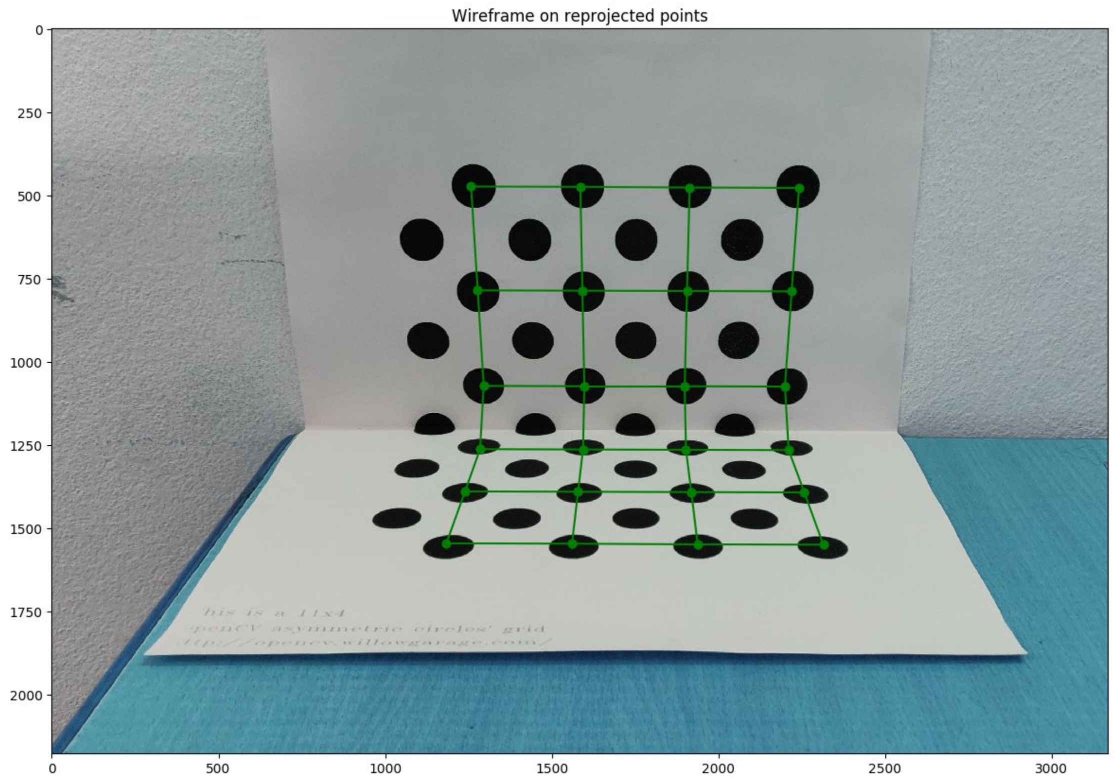
```
[ 7.72036157 20.68051947 49.56461806]
```





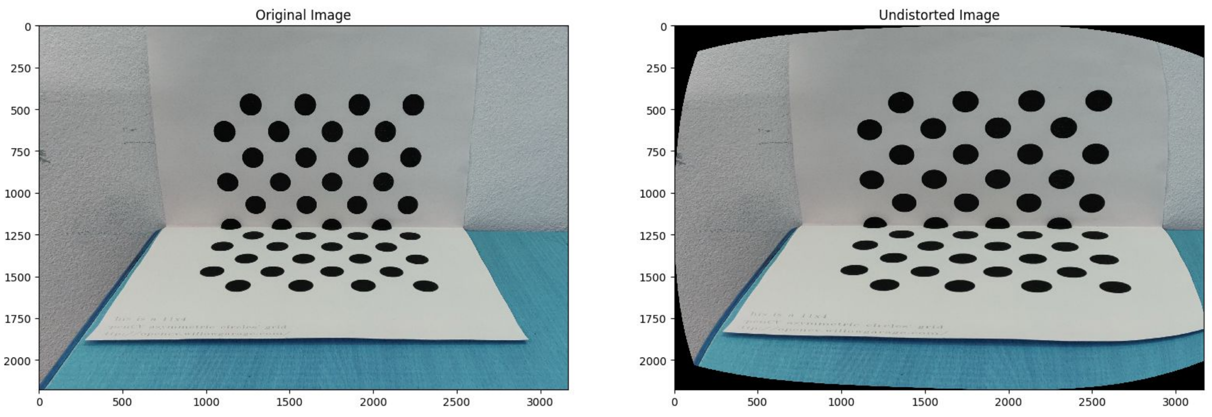
Blue Points: Original Points and Red Points: Reprojected Points

Wireframe Mesh

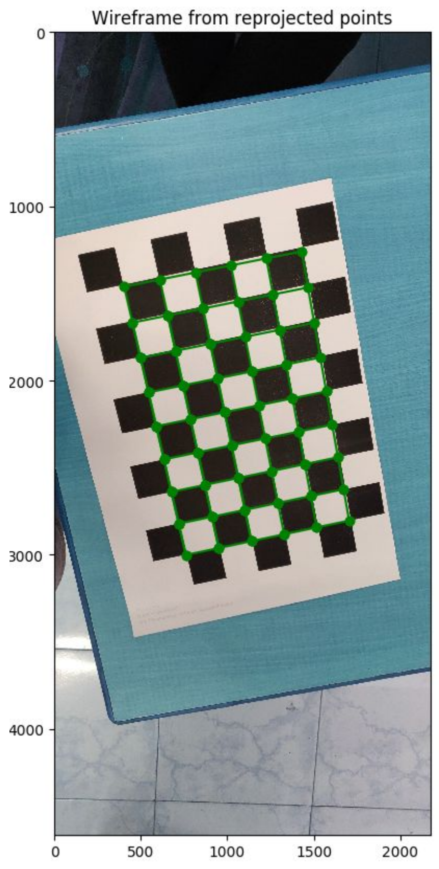


Distortion Parameters are:

```
[[ 1.18523079e+00 -1.36210618e+01 -7.03457368e-03 -6.96799172e-02
   6.24762948e+01]]
```



Zhang Method:
Wireframe

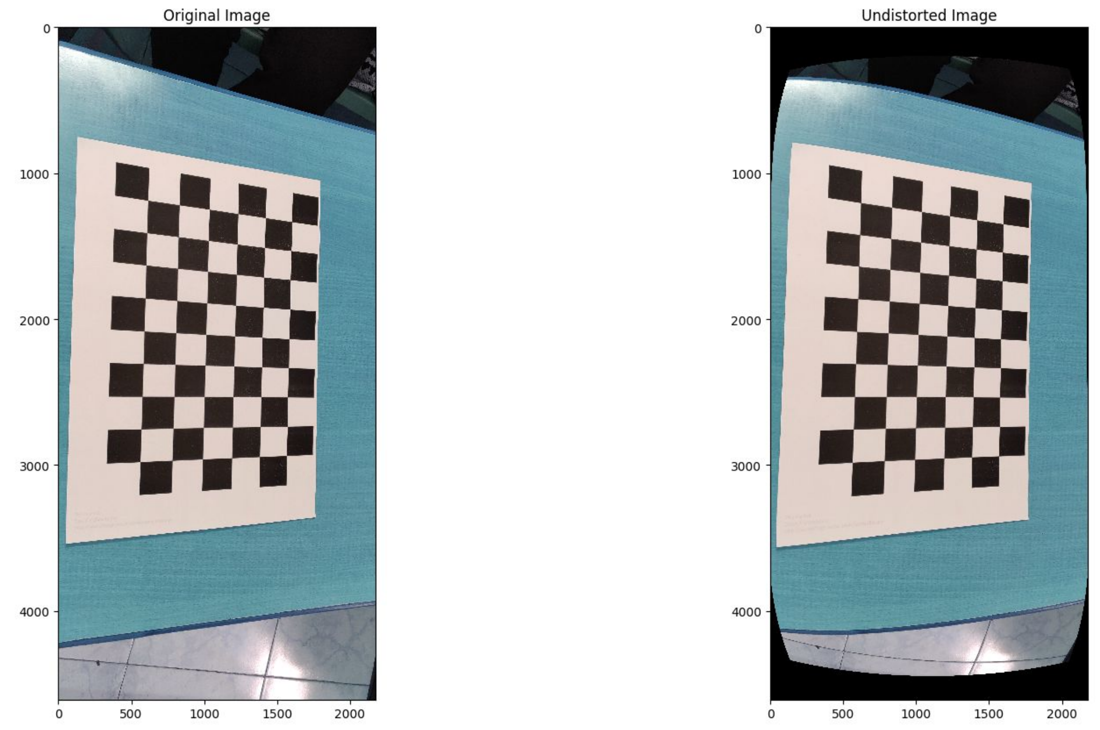


Camera Calibration Matrix:

```
[[3.56107491e+03 0.00000000e+00 1.11275328e+03]
 [0.00000000e+00 3.51899981e+03 2.28246940e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

Distortion Parameters:

```
[[ 1.96983916e-01 -2.36453158e+00 -1.37128214e-02  1.13648714e-03
   6.75362537e+00]]
```



As we can see that the camera with which photos were taken used focus and hence there were same variations observed in photos which lead to a bit variation from the original Projection Matrix.

Challenges Faced:

- As the focus of the phone changed, hence the Projection matrix for checkerboard using Zhang's Method was difficult to find.
- Due to radial distortions the original points and projected points were varying in some cases.
- There was a skew observed in the Camera Matrix obtained by DLT and RANSAC DLT.

Learnings:

- These experiments gave a lot of insight about all these variables and helped to understand them better.