Computer Vision Assignment 1

Camera Calibration

By- Apoorva Srivastava (2019702014)

1 Direct Linear Transform

1. For the given image calib-object.jpg using any 20-30 different points on different planes and perform the Direct Linear Transform (DLT) based calibration as discussed in class. Report the projection matrix, camera matrix, rotation matrix and projection center.
Note that you need to manually estimate the image co-ordinates of the given world points and refer to calib-object-legend.jpg for world measurements. Each chess block is 28X28 mm.

Theory:

Direct Linear Transform is a method to get the Intrinsic and Extrinsic Parameters of a Camera given we know at least 6, 3D-2D correspondences between the 3D world and the 2D image points. Below is the algorithm for implemeting the DLT-

Algorithm:

$$\begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

But we know only: $(x_i / w_i , y_i / w_i)$, Call it (u_i , v_i).

$$u_i = \frac{x_i}{w_i} = \frac{p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}}$$

$$v_i = \frac{y_i}{w_i} = \frac{p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}}$$

$$\begin{bmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & u_iX_i & u_iY_i & u_iZ_i & u_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & v_iX_i & v_iY_i & v_iZ_i & v_i \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ \vdots \\ p_{34} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \end{bmatrix}$$

- Stack equations for all points to get **Gp=0**.
- Solving this over-determined linear system of equations, we can recover the camera matrix.
- The matrix **P** can then be decomposed into the external and internal parameters: **K**, **R** and **t**.

Decomposing P

$$\mathbf{P} = \mathbf{K}\left[\mathbf{R} \mid \mathbf{t}\right]; \quad \mathbf{K} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

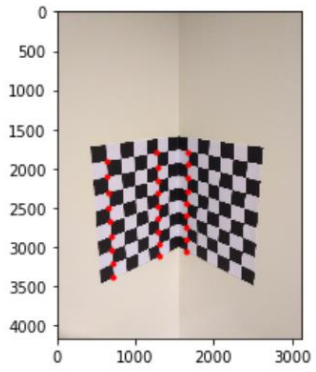
- Let $\mathbf{P}=[\mathbf{M}\ \mathbf{p}_4]$; where $\mathbf{M}=\mathbf{KR}$, and $\mathbf{p}_4=\mathbf{Kt}$.
- $\mathbf{MM}^T = \mathbf{KRR}^T\mathbf{K}^T = \mathbf{KK}^T$. We can solve for elements of **K**.
- $\mathbf{R} = \mathbf{K}^{-1}\mathbf{M}$, and $\mathbf{t} = \mathbf{K}^{-1}\mathbf{p}_4$.

Code:

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

img = plt.imread('calib-object.jpg')
pix=np.empty((27,2))
W=np.empty((27,3))
pix[:,0]=[1275,1279,1284,1285,1287,1289,1292,1298,1306,1664,1664,1661,1661,1658,1655,1656,1657,1657,629,
          642,655,655,677,689,697,708,717]
pix[:,1]=[1798,1978,2150,2315,2480,2640,2800,2956,3112,1783,1953,2119,2281,2441,2600,2752,2903,3055,1905,
          2106,2302,2494,2680,2862,3037,3212,3382]
W[:,0]=[0.056,0.056,0.056,0.056,0.056,0.056,0.056,0.056,0,0,0,0,0,0,0,0,0.168,0.168,0.168,0.168,
        0.168,0.168,0.168,0.168,0.168]
W[:,2]=[0,0,0,0,0,0,0,0,0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.028,0,0,0,0,0,0,0,0]
W[:,1]=[0.028,0.056,0.084,0.112,0.14,0.168,0.196,0.224,0.252,0.028,0.056,0.084,0.112,0.14,0.168,0.196,
        0.224,0.252,0.028,0.056,0.084,0.112,0.14,0.168,0.196,0.224,0.252]
plt.plot(pix[:,0],pix[:,1],'r.')

plt.imshow(img)
plt.show()
```




```
M=np.empty((54,12))
j=0
for i in range(27):
    M[j,:]=[W[i,0],W[i,1],W[i,2],1,0,0,0,0,-pix[i,0]*W[i,0],-pix[i,0]*W[i,1],-pix[i,0]*W[i,2],-pix[i,0]]
    M[j+1,:]=[0,0,0,0,W[i,0],W[i,1],W[i,2],1,-pix[i,1]*W[i,0],-pix[i,1]*W[i,1],-pix[i,1]*W[i,2],-pix[i,1]]
    j+=2
```

```
(U, S, V) = np.linalg.svd(M, full_matrices=True)
f = V[-1,:]
P =np.reshape(f,(3,4))
print(P)
```

```
[[ 6.23449449e-01 -5.95885817e-02 -2.76564655e-01 -1.55654252e-01]
 [ 1.20117711e-01 -6.77578070e-01  8.81040534e-02 -1.60440227e-01]
 [ 1.10874122e-04 -3.75012284e-05  1.02751371e-04 -1.01136527e-04]]
```

```
from scipy import linalg
KR=P[:,0:3]
K,R=linalg.rq(KR)
t=np.matmul(np.linalg.inv(K),P[:,-1])
K=K/K[2,2]
print(P)
print(K,R,t)
```

Projection Matrix is: [[6.23449449e-01, -5.95885817e-02, -2.76564655e-01, -1.55654252e-01]

[1.20117711e-01, -6.77578070e-01, 8.81040534e-02, -1.60440227e-01]

[1.10874122e-04, -3.75012284e-05, 1.02751371e-04, -1.01136527e-04]]

Camera Calibration Matrix is: [[-4.02307643e+03 -6.51346316e+01 1.77026024e+03]

[-0.00000000e+00 -3.99520617e+03 1.96975146e+03]

[-0.00000000e+00 -0.00000000e+00 1.00000000e+00]]

Rotation Matrix is: [[0.68430696, 0.02655791, -0.72871027]

[-0.1579395, -0.97021572, -0.18367519]

[-0.71188419, 0.24078235, -0.65973082]]

Translation Matrix is: [0.03631053, 0.06231262, 0.64936247]

```
projection_center=np.matmul(P,[0,0,1,1])
projection_center/=projection_center[2]
print('Image Projection Center is at:',projection_center)
```

```
Image Projection Center is at: [-2.67653676e+05 -4.47945301e+04  1.00000000e+00]
```

2. Implement the RANSAC based variant of the above calibration method and reportyour observations.

RANSAC Algorithm:

Determine:
  S — the smallest number of points required
  N — the number of iterations required
  d — the threshold used to identify a point that fits well
  T — the number of nearby points required
    to assert a model fits well
Until **N** iterations have occurred
  Draw a sample of **S** points from the data
    uniformly and at random
  Fit to that set of **S** points
  For each data point outside the sample
    Test the distance from the point to the line
      against **d** if the distance from the point to the line
      is less than **d** the point is close
  end
  If there are **T** or more points close to the line
    then there is a good fit. Refit the line using all
    these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion

Here we sampled 6 points from the available 27 correspondences and from those we calculated P matrix and we did this for 150 RANSAC iterations then we calculated the the min(avg(||x-PX||) for every iteration of RANSAC until the re-projection error was lesser than 3.5. So it was basically choosing the best P which suited all the point correspondences to get the camera matrix.

Code for DLT Implementation using RANSAC:

```
from scipy import linalg
RANSAC_iter=150
M=np.empty((12,12))
diff=np.empty((27,1))
num_points =[i for i in range(27)]
diff_norm=np.empty((RANSAC_iter,1))
P_=np.empty((RANSAC_iter,3,4))
norm=101
while norm>3.5:
    for k in range(RANSAC_iter): # RANSAC iterations
        six_p = sample(num_points, 6)
        #print(six_p)
        j=0
        for i in six_p:
            M[j,:]=[W[i,0],W[i,1],W[i,2],1,0,0,0,0,-pix[i,0]*W[i,0],-pix[i,0]*W[i,1],-pix[i,0]*W[i,2],-pix[i,0]]
            M[j+1,:]=[0,0,0,0,W[i,0],W[i,1],W[i,2],1,-pix[i,1]*W[i,0],-pix[i,1]*W[i,1],-pix[i,1]*W[i,2],-pix[i,1]]
            j+=2
        (U, S, V) = np.linalg.svd(M, full_matrices=True)
        f = V[-1,:]
        P_[k] =np.reshape(f,(3,4))
        for i in range(27):
            PX=np.matmul(P_[k],W[i])
            PX=PX/(PX[2]+0.0000001)
            diff[i]=np.linalg.norm(pix[i]-PX[0:2])
        diff_norm[k]=np.sum(diff)/27
    norm=np.min(diff_norm[:,0])
    #print(norm)
idx_min= np.argmin(diff_norm)
P=P_[idx_min]
print('diff_norm_chosen',diff_norm[idx_min])
print(P)
KR=P[:,0:3]
K,R=linalg.rq(KR)
t=np.matmul(linalg.pinv(K),P[:,-1])
K=K/K[2,2]
print('Projection Matrix is:',P)
print('Camera Calibration Matrix is:',K)
print('Rotation Matrix is:',R)
print('Translation Matrix is:',t)
```

The re-projection error is obtained by Calculating the min(avg(||x-PX||) for every iteration of RANSAC until the re-projection error was lesser than 3.5. The number of RANSAC iterations were 150.

Reprojection error threshold: 3.20308371

Projection Matrix is: [[6.19832731e-01, -5.89296718e-02, -2.92251715e-01, -1.55295655e-01]

[1.15940046e-01, -6.77861572e-01, 6.58043636e-02, -1.59806826e-01]

[1.09479051e-04, -3.80333416e-05, 9.53160973e-05, -1.00892050e-04]]

Camera Calibration Matrix is: [[-4.18106751e+03, -9.23015774e+01, 1.87605057e+03]

[-0.00000000e+00, -4.15290406e+03, 1.98720320e+03]

[-0.00000000e+00, -0.00000000e+00, 1.00000000e+00]]

Rotation Matrix is: [[0.66417231, 0.04113628, -0.74644688]

[-0.16306325, -0.96647137, -0.19835187]

[-0.729579, 0.25345787, -0.63519571]]

Translation Matrix is: [0.05272322, 0.06528844, 0.6723544]

The results obtained after DLT and RANSAC are quiet close, hence verifying their Correctness.

3. Repeat the above experiments after correcting for radial distortion. Estimate the radial distortion parameters from the straight lines in the image. What do you observe regarding the resulting parameters?

Theory about Distortion Coefficients:

Today's cheap pinhole cameras introduces a lot of distortion to images. Two major distortions are radial distortion and tangential distortion. Due to radial distortion, straight lines will appear curved. Its effect is more as we move away from the center of image.

This distortion is solved as follows:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Similarly, another distortion is the tangential distortion which occurs because image taking lense is not aligned perfectly parallel to the imaging plane. So some areas in image may look nearer than expected. It is solved as below:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

In short, we need to find five parameters, known as distortion coefficients given by:

$$Distortion\ coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Here we are obtaining these 5 parameter values for the given image using the inbuilt function cv2.CalibrateCamera() function.

The radial distortion parameters obtained are :

[[8.17420884e-01, -1.08688517e+01, -2.12979640e-02 , 2.60678703e-02, 6.70224886e+01]]

Code for Undistorting the images and then applying DLT and RANSAC:

```
#Obtaining the undistorted image
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = plt.imread('Fig1.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
pix=np.zeros((22,2),np.float32)
W=np.empty((22,3),np.float32)
K=np.array([[-4.02307643e+03, -6.51346316e+01, 960],
 [-0.00000000e+00, -3.99520617e+03,  616],
 [-0.00000000e+00, -0.00000000e+00,  1.00000000e+00]],np.float32)
temp_K = K.copy()
temp_K[ 0 , 1 ] = 0
temp_K[ 0 , 0 ] = abs(K[0,0])
temp_K[ 1 , 1 ] = abs(K[1,1])
objpoints=[]
imgpoints=[]
print(img.shape[1],img.shape[0])
pix[:,0]=[852,853,855,857,857,858,859,861,249,260,271,280,290,296,1233,1228,1225,1221,1220,1216,1218,1217]
pix[:,1]=[40,210,377,539,698,854,1007,1160,136,330,519,704,885,1061,65,241,413,581,746,907,1065,1219]
W[:,0]=[0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.14,0.14,0.14,0.14,0.14,0.14,0,0,0,0,0,0,0,0]
W[:,1]=[0,0.028,0.056,0.084,0.112,0.14,0.168,0.196,0,0.028,0.056,0.084,0.112,0.14,0,0.028,0.056,0.084,0.112,0.14,0.168,0.196]
W[:,2]=[0,0,0,0,0,0,0,0,0,0,0,0,0.056,0.056,0.056,0.056,0.056,0.056,0.056,0.056]
objpoints.append(W)
imgpoints.append(pix)

ret,mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, (img.shape[1],img.shape[0]),temp_K,None,None,
                                                  flags = (cv2.CALIB_USE_INTRINSIC_GUESS))
rotation_mat = np.zeros(shape=(3, 3))
R = cv2.Rodrigues(rvecs[0], rotation_mat)[0]
P = np.column_stack((np.matmul(mtx,R),np.matmul(mtx,tvecs[0])))
P=P/P[2,3]
print('Projection Matrix before undistortion is:',P)
print('Camera Calibration Matrix before undistortion is:',mtx)
print('Rotation Matrix before undistortion is:',R)

h,w = img.shape[: 2 ]
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h), 1 ,(w,h))
print( "The Radial Distortion Parameters are \n" ,dist)
# undistort
img_undistorted = cv2.undistort(img, mtx, dist, None , newcameramtx)
img_undist =img_undistorted*255
cv2.imwrite('./undist.jpg',img_undist)
plt.subplot( 121 )
plt.imshow(img)
plt.title( "Distorted Image" )

plt.subplot( 122 )
plt.imshow(img_undistorted)
plt.title( "Undistorted Image" )
plt.savefig('undistorted_im.jpg')
plt.show()
```

Projection Matrix before undistortion is:

[[-5.45141949e+03, 3.75017493e+02, 3.16853322e+03, 9.78394528e+02]

[7.03172010e+02, 5.96213700e+03, 7.66674871e+02, 2.04320008e+01]

[-1.10980146e+00 , 4.00652019e-01, -1.08620213e+00, 1.00000000e+00]]

Camera Calibration Matrix before undistortion is:

[[3.78973554e+03, 0.00000000e+00, 1.07253488e+03]

[0.00000000e+00, 3.76172113e+03, 3.01552130e+02]

[0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

Rotation Matrix before undistortion is:

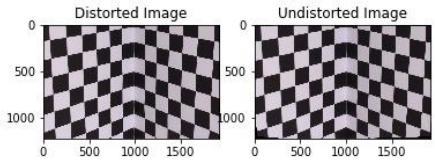
[[-0.7010966, -0.00899928, 0.71300951]

[0.17203026, 0.96824999, 0.18137681]

[-0.69200372, 0.24982188, -0.67728863]]

The Radial Distortion Parameters are:

[[8.17420884e-01, -1.08688517e+01, -2.12979640e-02 , 2.60678703e-02, 6.70224886e+01]]



Code for Applying DLT on the undistorted image:

```
# Implementation of DLT on undistorted image
import matplotlib.pyplot as plt
import numpy as np

img = plt.imread('undist.jpg')
pix=np.empty((22,2))
W=np.empty((22,3))
pix[:,0]=[862,862,864,867,866,866,867,868,253,260,271,279,288,295,1247,1245,1241,1237,1234,1229,1229,1226]
pix[:,1]=[40,209,377,540,700,856,1009,1160,140,333,522,708,889,1064,66,241,413,582,746,906,1062,1215]
W[:,0]=[0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.14,0.14,0.14,0.14,0.14,0.14,0,0,0,0,0,0,0]
W[:,1]=[0,0.028,0.056,0.084,0.112,0.14,0.168,0.196,0,0.028,0.056,0.084,0.112,0.14,0,0.028,0.056,0.084,0.112,0.14,0.168,0.196]
W[:,2]=[0,0,0,0,0,0,0,0,0,0,0,0,0,0.056,0.056,0.056,0.056,0.056,0.056,0.056,0.056]
plt.plot(pix[:,0],pix[:,1],'r.')

plt.imshow(img)
plt.show()

M=np.empty((44,12))
j=0
for i in range(22):
    M[j,:]=[W[i,0],W[i,1],W[i,2],1,0,0,0,0,-pix[i,0]*W[i,0],-pix[i,0]*W[i,1],-pix[i,0]*W[i,2],-pix[i,0]]
    M[j+1,:]=[0,0,0,0,W[i,0],W[i,1],W[i,2],1,-pix[i,1]*W[i,0],-pix[i,1]*W[i,1],-pix[i,1]*W[i,2],-pix[i,1]]
    j+=2

(U, S, V) = np.linalg.svd(M, full_matrices=True)
f = V[-1,:]
P =np.reshape(f,(3,4))
print(P)

from scipy import linalg
KR=P[:,0:3]
K,R=linalg.rq(KR)
t=np.matmul(np.linalg.inv(K),P[:,-1])
K=K/K[2,2]
print('Projection Matrix after undistortion and performing DLT is:',P)
print('Camera Calibration Matrix after undistortion and performing DLT is:',K)
print('Rotation Matrix after undistortion and performing DLT is:',R)
print('Translation vector after undistortion and performing DLT is:',t)
```

Projection Matrix after undistortion and performing DLT is:

[[6.22428991e-01, -4.18347147e-02, -3.64991116e-01, -1.11029347e-01]

[-7.94980313e-02, -6.71669576e-01, -8.84415785e-02, -2.01587695e-03]

[1.25306303e-04, -4.40138935e-05, 1.19985601e-04, -1.12105574e-04]]

Camera Calibration Matrix after undistortion and performing DLT is:

[[-3.87823552e+03, -6.43814108e+00, 1.12506367e+03]

[-0.00000000e+00, -3.80068841e+03, 2.80610276e+02]

[-0.00000000e+00, -0.00000000e+00, 1.00000000e+00]]

Rotation Matrix after undistortion and performing DLT is:

[[0.69387004, 0.0126783, -0.71998863]

[-0.16855256, -0.96920999, -0.17950493]

[-0.70009599, 0.24590902, -0.67036882]]

Translation vector after undistortion and performing DLT is:

[0.02167661, 0.04328039, 0.6263425]

Code for Implementing RANSAC on Undistorted image:

```
# RANSAC Implementation after Distortion
import matplotlib.pyplot as plt
import numpy as np
from random import sample
from scipy import linalg

img = plt.imread('undist.jpg')
pix=np.empty((22,2))
W=np.ones((22,4))
pix[:,0]=[862,862,864,867,866,866,867,868,253,260,271,279,288,295,1247,1245,1241,1237,1234,1229,1229,1226]
pix[:,1]=[40,209,377,540,700,856,1009,1160,140,333,522,708,889,1064,66,241,413,582,746,906,1062,1215]
W[:,0]=[0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.028,0.14,0.14,0.14,0.14,0.14,0.14,0.14,0,0,0,0,0,0,0,0]
W[:,1]=[0,0.028,0.056,0.084,0.112,0.14,0.168,0.196,0,0.028,0.056,0.084,0.112,0.14,0,0.028,0.056,0.084,0.112,0.14,0.168,0.196]
W[:,2]=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.056,0.056,0.056,0.056,0.056,0.056,0.056,0.056]
plt.plot(pix[:,0],pix[:,1],'r.')
plt.imshow(img)
plt.show()
RANSAC_iter=150
M=np.empty((12,12))
diff=np.empty((22,1))
num_points =[i for i in range(22)]
diff_norm=np.empty((RANSAC_iter,1))
P_=np.empty((RANSAC_iter,3,4))
norm=101
while norm>3.5:
    for k in range(RANSAC_iter): # RANSAC iterations
        six_p = sample(num_points, 6)
        #print(six_p)
        j=0
        for i in six_p:
            M[j,:]=[W[i,0],W[i,1],W[i,2],1,0,0,0,0,-pix[i,0]*W[i,0],-pix[i,0]*W[i,1],-pix[i,0]*W[i,2],-pix[i,0]]
            M[j+1,:]=[0,0,0,0,W[i,0],W[i,1],W[i,2],1,-pix[i,1]*W[i,0],-pix[i,1]*W[i,1],-pix[i,1]*W[i,2],-pix[i,1]]
            j+=2
        (U, S, V) = np.linalg.svd(M, full_matrices=True)
        f = V[-1,:]
        P_[k] =np.reshape(f,(3,4))
        for i in range(22):
            PX=np.matmul(P_[k],W[i])
            PX=PX/(PX[2]+0.0000001)
            diff[i]=np.linalg.norm(pix[i]-PX[0:2])
        diff_norm[k]=np.sum(diff)/22
    norm=np.min(diff_norm[:,0])
    #print(norm)
idx_min= np.argmin(diff_norm)
P=P_[idx_min]
print('diff_norm_chosen',diff_norm[idx_min])
#print(P)
KR=P[:,0:3]
K,R=linalg.rq(KR)
t=np.matmul(linalg.pinv(K),P[:,-1])
K=K/K[2,2]
print('Projection Matrix after undistortion and performing DLTis:',P)
print('Camera Calibration after undistortion and performing DLT Matrix is:',K)
```

Reprojection Error =1.56177285

Projection Matrix after undistortion and performing DLT is:

[[-6.20076782e-01, 4.23143901e-02, 3.68750961e-01, 1.10721397e-01]

[8.13356654e-02, 6.71446787e-01, 8.95435808e-02, 1.84472873e-03]

[-1.24328977e-04, 4.43177850e-05, -1.19413730e-04, 1.11918059e-04]]

Camera Calibration after undistortion and performing DLT Matrix is:

[[-3.90753468e+03, -9.29400051e+00, 1.10269322e+03]

[0.00000000e+00, -3.82263171e+03, 2.82560788e+02]

[0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

Rotation Matrix after undistortion and performing DLT is:

[[0.69483027, 0.01172744, -0.71907814]

 [-0.17117294, -0.96843594, -0.18119507]

 [-0.69850607, 0.24898654, -0.67089118]]

Translation Matrix after undistortion and performing DLT is: [0.01814127, 0.04376676, 0.62877895]

Observations:

The reprojection error reduced after undistortion of the image in both the cases but skew parameter increased after the undistortion.

After the undistortion the straight lines were appearing instead of the slight bulge.

2. Zhangs Method

1. Use checkerboard images IMG5456.JPG - IMG5470.JPG and perform camera calibration using Zhang's Method. You can use the available OpenCV or Matlab implementation.

2. Using the estimated camera parameters compute the image points and overlay a wire-frame over the actual image of chessboard using straight lines between the computed points. Refer to example-wireframe.png for reference. What do you observe about the overlay? Note that do not use the image points found using the cvFindChess-boardCorners for wireframe overlay. Size of each square on checkerboard 29mmX29mm

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

Theory:

Zhang’s Method is a 2D Camera Calibration method which uses 2D-2D Homography matrix to calculate the Camera Parameters.

Code:

```
import numpy as np
import cv2
import glob
import matplotlib.pyplot as plt
# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*8,3), np.float32)
objp[:,:2] = np.mgrid[0:8,0:6].T.reshape(-1,2)
# 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('*.jpg')
for i in range(56,71,1):
    img = cv2.imread('IMG_54'+str(i)+'.jpg')
    #print('IMG_54'+str(i)+'.jpg')
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (8,6),None)
    #print(ret)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (8,6), corners2,ret)
        cv2.imwrite('img'+str(i)+'.jpg',img)
        #cv2.imshow('img',img)
        #cv2.waitKey(500)

cv2.destroyAllWindows()
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)
print(mtx)
```

```
rotation_mat = np.zeros(shape=(3, 3))
projected_points=np.empty((48,3))
R = cv2.Rodrigues(rvecs[0], rotation_mat)[0]
P = np.column_stack((np.matmul(mtx,R),np.matmul(mtx,tvecs[0])))
P=P/P[2,3]
print('Projection Matrix is:',P)
print('Camera Calibration Matrix is:',mtx)
print('Rotation Matrix ',R)

for i in range(48):
    np.append(objpoints[0][i],1)
    projected_points[i]=np.matmul(P,np.append(objpoints[0][i],1))
    projected_points[i]=projected_points[i]/projected_points[i,2]

im=plt.imread('IMG_5456.jpg')
plt.plot(projected_points[:,0],projected_points[:,1],'ro')
plt.title('Reprojected_Points')
plt.imshow(im)
plt.show()
```

Projection Matrix is:

[[4.46765428e+02, 3.48582750e-01, 8.65198264e+01, 1.06140982e+03]

[1.05129517e+00, 4.42363080e+02, 5.32162630e+01, 5.88039323e+02]

[1.52972824e-03, -3.34831460e-04, 3.23769470e-02, 1.00000000e+00]]

Camera Calibration Matrix is:

[[1.36415094e+04, 0.00000000e+00, 3.31635881e+03]

[0.00000000e+00, 1.36632517e+04, 1.50037396e+03]

[0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

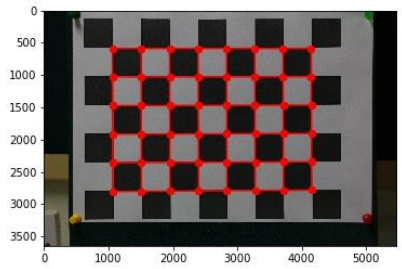
Rotation Matrix :

[[0.99888188, 0.00329952, -0.0471605]

[-0.00280852, 0.9999412,   0.01047375]

[0.04719229, -0.01032959, 0.99883241]]

```
### Wireframe
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]],'ro-')
for i in range(9):
    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]],'ro-')
        i1 = j
        j = j + 8
plt.imshow(im)
plt.savefig('Wireframe.jpg')
plt.title('Wireframe from reprojected points')
plt.show()
```



Observation about Overlay:

The overlay is slightly deviated from the actual lines of the chessboard and this is due to Reprojection error in the reprojected points. Rest the frame is fitting perfectly with the reprojected points.Parallel points form parallel lines.

The World origin O=[0,0,0] will project at the pixel value =P*O where P is the projection matrix. This bears out in my observation as it maps to the pixel location corresponding to which the world point was taken to be World Origin.

```
print('The world origin is:',objpoints[0][0])
print('The projection of world origin is:',projected_points[0,:2],'pixels')
```

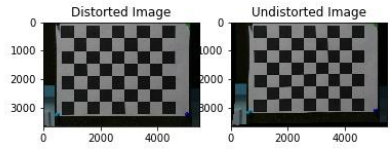
```
The world origin is: [0. 0. 0.]
The projection of world origin is: [1061.40981578  588.03932323] pixels
```

The planar image provided was undistorted using the following code:

```
# Undistorting the image

im = cv2.imread('IMG_5456.jpg')
h,w = im.shape[: 2 ]
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h), 1 ,(w,h))
print( "The Radial Distortion Parameters are \n" ,dist)
# undistort
img_undistorted = cv2.undistort(im, mtx, dist, None , newcameramtx)
plt.subplot( 121 )
plt.imshow(im)
plt.title( "Distorted Image" )
plt.subplot( 122 )
plt.imshow(img_undistorted)
plt.title( "Undistorted Image" )
plt.savefig('Planar_Undistort.jpg')
```

```
The Radial Distortion Parameters are
 [[ 9.79057908e-02  9.45876425e+00 -1.53012034e-02  2.72096493e-02
  -1.48434591e+02]]
```



3. Hands on

Select a camera that you would like to use for the assignments. Note that you might be using this camera for future assignments also.

1. Perform the above calibration methods using the images taken by your camera. Use the calibration object for which you can measure the world co-ordinates for DLT and printed checkerboard pattern for Zhangs Method.
2. Vary the focus of your phone and comment on the results.

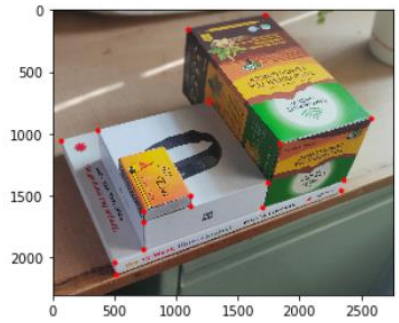
Code for performing DLT on self created 3D:

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

img = plt.imread('IMG_1.jpg')
pix=np.empty((18,2))
W=np.empty((18,3))
pix[:,0]=[514,2332,503,737,1696,2360,733,1114,1744,730,1119,1833,2575,360,1260,70,1089,1723]
pix[:,1]=[2137,1461,2042,1935,1594,1375,1710,1586,1394,1623,1497,1076,871,975,739,1053,155,39]

W[:,0]=[0.197,0,0.197,0.177,0.08,0,0.177,0.142,0.08,0.177,0.142,0.08,0,0.177,0.08,0.197,0.08,0]
W[:,1]=[0,0,0.014,0.014,0.014,0.014,0.046,0.046,0.046,0.057,0.057,0.082,0.082,0.046,0.046,0.014,0.082,0.082]
W[:,2]=[0,0,0,0,0,0,0,0,0,0,0,0,-0.097,-0.097,-0.13,-0.15,-0.15]
plt.plot(pix[:,0],pix[:,1],'r.')

plt.imshow(img)
plt.show()
```




```
M=np.empty((36,12))
j=0
for i in range(18):
    M[j,:]=[W[i,0],W[i,1],W[i,2],1,0,0,0,0,-pix[i,0]*W[i,0],-pix[i,0]*W[i,1],-pix[i,0]*W[i,2],-pix[i,0]]
    M[j+1,:]=[0,0,0,0,W[i,0],W[i,1],W[i,2],1,-pix[i,1]*W[i,0],-pix[i,1]*W[i,1],-pix[i,1]*W[i,2],-pix[i,1]]
    j+=2
(U, S, V) = np.linalg.svd(M, full_matrices=True)
f = V[-1,:]
P =np.reshape(f,(3,4))
#print(P)
```

```
from scipy import linalg
KR=P[:,0:3]
K,R=linalg.rq(KR)
t=np.matmul(np.linalg.inv(K),P[:,-1])
K=K/K[2,2]
print('Projection Matrix is',P)
print('Camera Calibration Matrix is',K)
print('Rotation Matrix is',R)
print('Translation Vector is',t)
```

Projection Matrix is:

[[-6.66412942e-01,-7.72228885e-02, 1.77650895e-01, 1.60522442e-01]

[1.04222236e-01, -6.05392839e-01, 3.24064272e-01, 1.01107248e-01]

[-5.95999861e-05, -1.09722742e-04, -9.13745265e-05, 6.90066296e-05]]

Camera Calibration Matrix is :

[[-4.28163083e+03, -5.81107041e+01, 1.33491306e+03]

[0.00000000e+00, 4.30292050e+03, 1.27827090e+03]

[0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

Rotation Matrix is:

[[0.88215663 -0.0950448 -0.46126583]

[0.27097141 -0.69863605 0.66217986]

[-0.38519369 -0.7091362 -0.590552]]

Translation Vector is [-0.10351744 0.01937291 0.44598867]

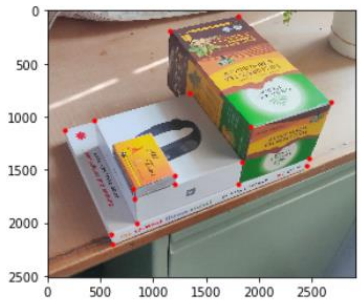
Code for performing DLT after varying focus:

```
#Varying the focus
import matplotlib.pyplot as plt
import numpy as np

img = plt.imread('IM_1.jpg')
pix=np.empty((18,2))
W=np.empty((18,3))
pix[:,0]=[619,2446,602,844,1803,2473,820,1204,1839,814,1201,1921,2677,442,1341,160,1158,1800]
pix[:,1]=[2202,1467,2107,2000,1629,1387,1770,1631,1427,1679,1548,1097,862,1040,778,1124,191,54]


W[:,0]=[0.197,0,0.197,0.177,0.08,0,0.177,0.142,0.08,0.177,0.142,0.08,0,0.177,0.08,0.197,0.08,0]
W[:,1]=[0,0,0.014,0.014,0.014,0.014,0.046,0.046,0.046,0.057,0.057,0.082,0.082,0.046,0.046,0.014,0.082,0.082]
W[:,2]=[0,0,0,0,0,0,0,0,0,0,0,0,-0.097,-0.097,-0.13,-0.15,-0.15]
plt.plot(pix[:,0],pix[:,1],'r.')

plt.imshow(img)
plt.show()
```



```
M=np.empty((36,12))
j=0
for i in range(18):
    M[j,:]=[W[i,0],W[i,1],W[i,2],1,0,0,0,0,-pix[i,0]*W[i,0],-pix[i,0]*W[i,1],-pix[i,0]*W[i,2],-pix[i,0]]
    M[j+1,:]=[0,0,0,0,W[i,0],W[i,1],W[i,2],1,-pix[i,1]*W[i,0],-pix[i,1]*W[i,1],-pix[i,1]*W[i,2],-pix[i,1]]
    j+=2
```

```
(U, S, V) = np.linalg.svd(M, full_matrices=True)
f = V[-1,:]
P =np.reshape(f,(3,4))
#print(P)
```

```
from scipy import linalg
KR=P[:,0:3]
K,R=linalg.rq(KR)
t=np.matmul(np.linalg.inv(K),P[:,-1])
K=K/K[2,2]
print(P)
print(K,R,t)
```

Projection Matrix is

[[-6.62062157e-01, -1.12491237e-01, 1.69924601e-01, 1.65242404e-01]

 [1.20576360e-01, -6.11755094e-01, 3.06694057e-01, 9.99788538e-02]

 [-5.68404739e-05, -1.13247833e-04, -9.25579164e-05, 6.76963638e-05]]

Camera Calibration Matrix is

[[-4.18381846e+03, -6.55405144e+01, 1.40696173e+03]

 [0.00000000e+00, 4.20695470e+03, 1.38242805e+03]

 [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

Rotation Matrix is

 [[0.88191488, -0.06055121, -0.46750368]

 [0.301684,   -0.68954593, 0.65841717]

 [-0.36223322, -0.72170628, -0.58985349]]

Translation Vector is [-0.10677015, 0.00968504, 0.43141568]

Zhang's HandsOn Code:

```
import numpy as np
import cv2
import glob

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*9,3), np.float32)
objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
# 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.
rotation_mat = np.zeros(shape=(3, 3))
#images = glob.glob('*.jpg')
for i in range(1,18,1):
    img = cv2.imread('./HandsOn/IMG_'+str(i)+'.jpg')
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (9,6),None)
    #print(ret)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (9,6), corners2,ret)
        cv2.imwrite('img'+str(i)+'.jpg',img)
        #cv2.imshow('img',img)
        #cv2.waitKey(500)

cv2.destroyAllWindows()
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)
R = cv2.Rodrigues(rvecs[0], rotation_mat)[0]
P = np.column_stack((np.matmul(mtx,R),np.matmul(mtx,tvecs[0])))
P=P/P[2,3]
```

```
print('Camera Matrix:',mtx)
print('Rotaion Matrix:',R)
print('Projection Matrix:',P)
```

Camera Matrix:

[[3.95667606e+03, 0.00000000e+00, 7.10247209e+02]

[0.00000000e+00, 3.88535462e+03, 1.05592942e+02]

[0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

Rotaion Matrix: [[-0.57797587, 0.52617304, -0.62376744]

[-0.68532909, -0.72793127, 0.02097876]

[-0.44302137, 0.43961119, 0.7813284]]

Projection Matrix: [[-1.36727181e+02, 1.25827497e+02, -1.00546703e+02, 1.38388797e+03]

[-1.42403750e+02, -1.46204903e+02, 8.61996341e+00, 2.21644402e+03]

[-2.32837381e-02, 2.31045102e-02, 4.10640367e-02, 1.00000000e+00]]

```python
#After Changing the focus
import numpy as np
import cv2
import glob

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*9,3), np.float32)
objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
# 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.
rotation_mat = np.zeros(shape=(3, 3))
#images = glob.glob('*.jpg')
for i in range(1,18,1):
    img = cv2.imread('./HandsOn/IM_'+str(i)+'.jpg')

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

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (9,6),None)
    #print(ret)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        img = cv2.drawChessboardCorners(img, (9,6), corners2,ret)
        cv2.imwrite('img'+str(i)+'.jpg',img)
        #cv2.imshow('img',img)
        #cv2.waitKey(500)

cv2.destroyAllWindows()
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1],None,None)
R = cv2.Rodrigues(rvecs[0], rotation_mat)[0]
P = np.column_stack((np.matmul(mtx,R),np.matmul(mtx,tvecs[0])))
P=P/P[2,3]
```

```python
print('Camera Matrix:',mtx)
print('Rotation Matrix:',R)
#print('Translation Vector',tvecs)
print('Projection Matrix:',P)
```

Camera Matrix:

[[3.97399370e+03, 0.00000000e+00, 1.49676828e+03]

[0.00000000e+00, 4.04303442e+03, 4.80908369e+02]

[0.00000000e+00, 0.00000000e+00, 1.00000000e+00]]

Rotation Matrix:

[[-0.9816235, -0.14069016, -0.12892471]

[0.01677175, -0.7365994, 0.67612132]

[-0.19008948, 0.66153428, 0.7254229]]

Projection Matrix:

[[-2.54769234e+02, 2.62385954e+01, 3.49053627e+01, 2.43468858e+03]

[-1.43694200e+00, -1.61910918e+02, 1.87627414e+02, 1.06567677e+03]

[-1.15706892e-02, 4.02673913e-02, 4.41562719e-02, 1.00000000e+00]]

Observations:

On changing the focus, in both the case of DLT and Zhang's method, the value of focal length in the camera matrix changed. Also the image center varied on changed on changing the focus. The Camera Matrix obtained after performing DLT and Zhang's are quiet close,hence verifying their authenticity.

Link for Self-Clicked 3D images, ChessBoard image folder in different focuses and the images of intermediate results:

https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/EqEpe-SBKb5Au820NsV9NcYBOtna6nkqhAyoQfkZmJXH-Q?e=heeRqn