CV13

February 3, 2019

1 Computer Vision Assignment 1

1.1 Kanav Gupta

1.2 Roll No. 20161151

```
In [502]: import cv2
          import matplotlib.pyplot as plt
          import numpy as np
          from glob import glob
          import math
          import random
          from PIL import Image
          import pdb
```

1.3 KRT

Finds Camera Matrix, Rotation Matrix and Camera Centre QR decomposition of P gives us K and R which are further used to obtain T (Camera Center)

```
In [503]: def KRT(P):
              t1 = P[0:3,0:3]
              t2 = P[:3,3]
              TM = -1*np.matmul(np.linalg.inv(t1),t2)
              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]
              return K,R,TM
```

1.4 DLT

Calculates Projection matrix using Direct Linear Transformation method.

First matrix of size (2*n X 12) is created and provided with the required entities. Last row of SVD of this matrix is taken and reshaped into projection matrix

```
In [504]: def DLT(world_pts,img_points):
              n = world_pts.shape[0]
```

```
    A =  np.zeros((2*n,12))
    for i in range(n):
        A[i*2,0:3] = -1 * world_pts[i,:]
        A[i*2,3] = -1
        A[i*2,8:11] = img_points[i,0]*world_pts[i,:]
        A[i*2,11] = img_points[i,0]
        A[i*2+1,4:7] = -1 * world_pts[i,:]
        A[i*2+1,7] = -1
        A[i*2+1,8:11] = img_points[i,1]*world_pts[i,:]
        A[i*2+1,11] = img_points[i,1]
    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
```

1.5 MSE

Calculates error between actual and reprojected point

```
In [505]: def mse(projected_pts, image_pts):
              x = projected_pts[0] - image_pts[0]
              y = projected_pts[1] - image_pts[1]
              return np.abs(x) + np.abs(y)
```

1.6 reprojection_error

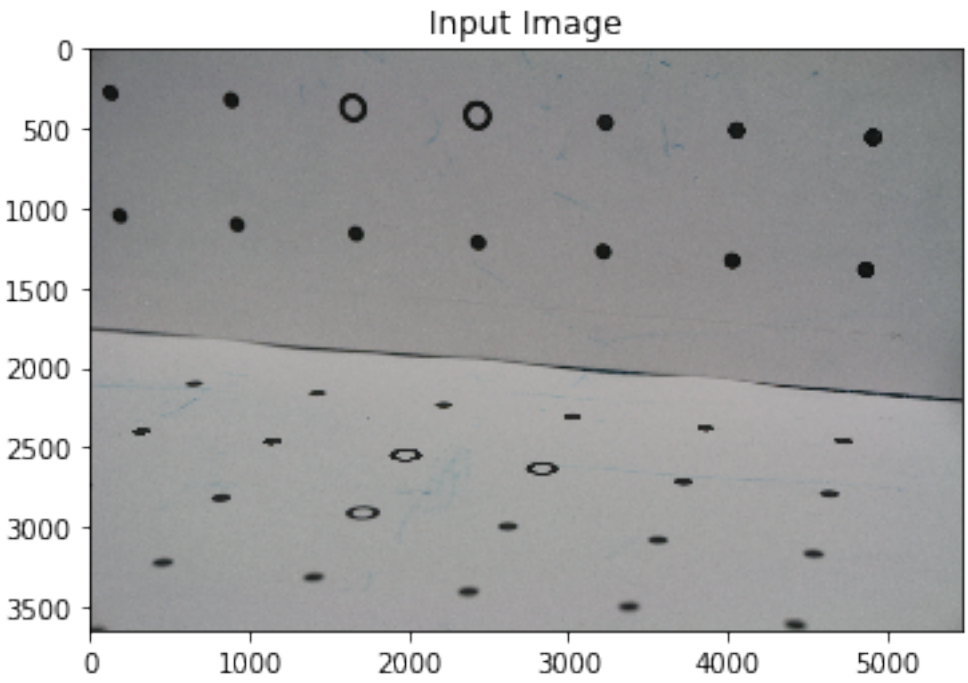
Returns total inliers points (within theta range) for specific projection matrix.

```
In [506]: def reprojection_error(P,image_points,world_points):
              theta = 1.5
              nn = image_points.shape[0]
              W = np.zeros((nn,4))
              W[:,0:3] = world_points[:,0:3]
              W[:,3] = 1
              inliers = 0
              for i in range(nn):
                  projected_points = np.matmul(P,np.transpose(W[i,:]))
                  projected_points =  projected_points/projected_points[2]
                  if mse(projected_points,image_points[i]) < theta:
                      inliers+=1
              return inliers
```

1.7 RANSAC

Takes 6 random points in each of N iterations. Calculates projection matrix using DLT function described above using 6 chosen points Calculate inliers for that projection matrix Returns projection matrix which has most number of inliers (calculated from above function)

```
In [507]: def RANSAC(img_points,world_pts):
              N = 10000
              n = img_points.shape[0]
              image_points = np.zeros((n-6,2))
              world_points = np.zeros((n-6,3))
              proj_imps = np.zeros((6,2))
              proj_wrps = np.zeros((6,3))
              best_projection_matrix = []
              inliers_max = 0
              for i in range(N):
                  l  = random.sample(range(n),6)
                  p = 0
                  q = 0
                  for j in range(n):
                      if j in l:
                          proj_imps[q,:] = img_points[j,:]
                          proj_wrps[q,:] = world_pts[j,:]
                          q = q +1
                      else:
                          image_points[p,:] = img_points[j,:]
                          world_points[p,:] = world_pts[j,:]
                          p = p+1
                  if (((np.sum(proj_wrps,axis=1))[1] == 0) or ((np.sum(proj_wrps,axis=1))[2] ==
                      continue
                  P = DLT(proj_wrps,proj_imps)
                  inliers = reprojection_error(P,image_points,world_points)
                  if (inliers_max < inliers):
                      best_projection_matrix =  P.copy()
                      inliers_max = inliers
                      print('Max Inlier Update to ----> ' + str(inliers_max))
              return best_projection_matrix

In [508]: I = cv2.imread('data/IMG_5455.JPG')
          plt.imshow(I)
          plt.title('Input Image')
          plt.show()
```



Defining Image Points and corresponding World coordinated

```
In [509]: X = [216,180,144,108,72,36,0,216,180,144,108,72,36,0,180,144,108,72,36,0,180,144,108,7
          Y = [72,72,72,72,72,72,72,36,36,36,36,36,36,36,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
          Z = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,36,36,36,36,36,36,72,72,72,72,72,72,108,108,108,108,1
          img_points  = [[140,285],[908,338],[1655,391],[2450,431],[3223,465],[4070,535],[4945,5
                     [2456,1232],[3243,1265],[4064,1326],[4878,1392],[661,2100],[1435,2166],[222
                    [4744,2460],[334,2400],[1148,2460],[1995,2547],[2849,2627],[3737,2713],[4658
                    ,[3570,3080],[4551,3167],[474,3214],[1408,3314],[2389,3401],[3377,3488],[443
          n1 = len(Y)
          world_pts = []
          for i in range(n):
              world_pts.append([X[i],Y[i],Z[i]])
          img_points = np.array(img_points)
          world_pts = np.array(world_pts)
```

1.8 Finding Projection Matrix using DLT

```
In [510]: P1 = DLT(world_pts,img_points)
          print('Projection matrix')
          print(P1)
          krt = KRT(P1)
          print('Camera Matrix')
          print(krt[0])
          print('Rotation Matrix')
          print(krt[1])
```

```
print('Camera Centre')
print(krt[2])
```

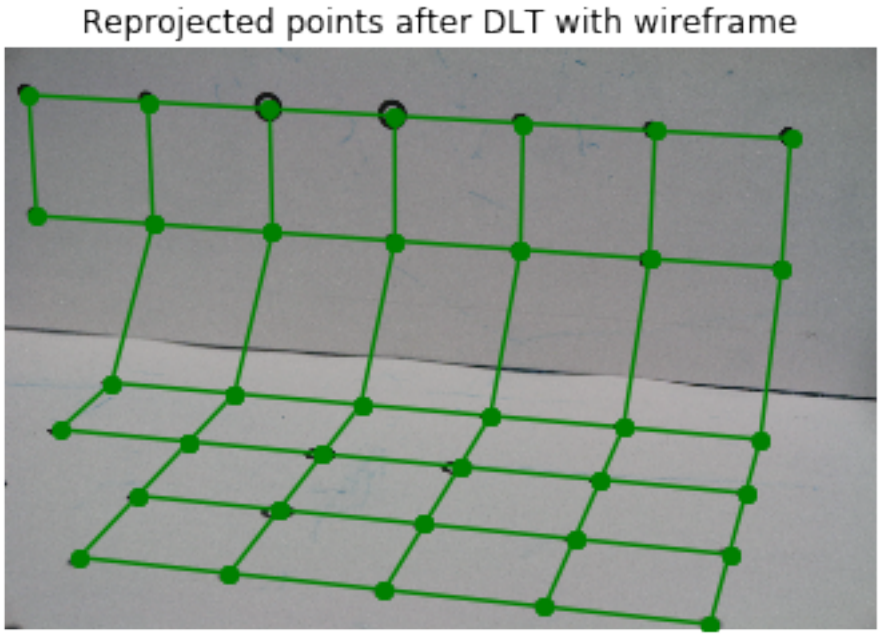
```
Projection matrix
[[ -2.11225340e+01  -1.52511970e+00  -9.72155998e+00   4.82458631e+03]
 [ -1.09832879e+00  -2.24096507e+01   4.43993421e+00   2.15758505e+03]
 [  3.60153742e-04  -6.15323259e-04  -1.57382436e-03   1.00000000e+00]]
Camera Matrix
[[ -1.31727418e+04   1.10459423e+02   2.89124562e+03]
 [ -0.00000000e+00  -1.30624295e+04   2.14585760e+03]
 [ -0.00000000e+00  -0.00000000e+00   1.00000000e+00]]
Rotation Matrix
[[-0.9745131   0.00332142 -0.22430602]
 [-0.08290827 -0.93442929  0.34636416]
 [-0.2084477   0.35613323  0.91088895]]
Camera Centre
[ -37.81612897  206.32412608  546.07386058]
```

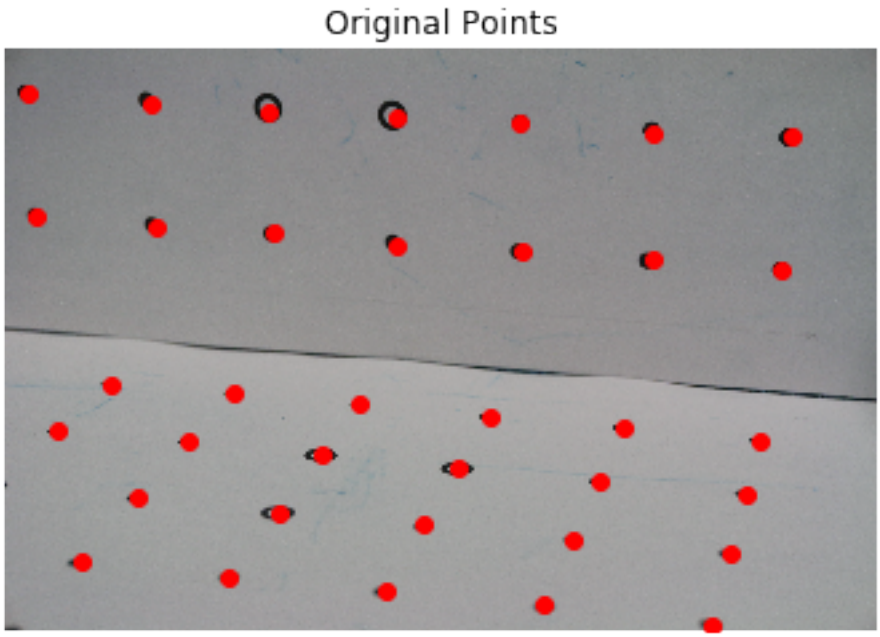
In [511]:
```
n = img_points.shape[0]
W = np.zeros((n,4))
W[:,0:3] = world_pts[:,0:3]
W[:,3] = 1
pp1 = np.zeros((n,3))
for i in range(n):
    pp1[i,:] = np.matmul(P1,np.transpose(W[i,:]))
    pp1[i,:] =  pp1[i,:]/pp1[i,2]

ppp1 = []
for aa in pp1:
    ppp1.append([int(aa[0]),int(aa[1])])
print('Reprojected points after DLT:')
print(ppp1)
```

```
Reprojected points after DLT:
[[147, 296], [894, 339], [1660, 383], [2446, 427], [3253, 473], [4082, 520], [4933, 569], [196,
```

In [512]:
```
reim = I.copy()
plt.imshow(reim)
plt.title('Reprojected points after DLT with wireframe')
plt.axis('off')

pts = [7,14,20,26,31,36]

q = 0
for i in range(35):
    if i == pts[q]-1:
        q+=1
```

```
            continue
        plt.plot([ppp1[i][0],ppp1[i+1][0]],[ppp1[i][1],ppp1[i+1][1]],'go-')
idx = [7,6,6,5,5,4]
num = [2,4,6,6,6,6,6]

for i in range(7):
    prev = i
    for j in range(num[i]-1):
        nxt = prev + idx[j]
        plt.plot([ppp1[prev][0],ppp1[nxt][0]],[ppp1[prev][1],ppp1[nxt][1]],'go-')
        prev = nxt

plt.show()

reim1 = I.copy()
plt.imshow(reim1)
plt.title('Original Points')
plt.axis('off')

plt.plot(img_points[:,0],img_points[:,1],'ro')
plt.show()
```





1.9 Calibration using RANSAC Method

Numbers below denote inliers update (rise) Projection Matrix P is obtained as output

```
In [513]: P = RANSAC(img_points,world_pts)
          print('Projection Matrix')
          print(P)
```

```
Max Inlier Update to ----> 1
Max Inlier Update to ----> 2
```

```
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:17: RuntimeWarning: divide by zero
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:17: RuntimeWarning: invalid value e
```

```
Max Inlier Update to ----> 3
Projection Matrix
[[ -2.11233718e+01  -1.47920775e+00  -9.60966742e+00   4.81085727e+03]
 [ -1.13868831e+00  -2.23424576e+01   4.55383210e+00   2.15005910e+03]
 [  3.38272477e-04  -7.94844038e-04  -1.57582044e-03   1.00000000e+00]]
```

```
In [514]: X = KRT(P)
          print('Camera Marix')
```

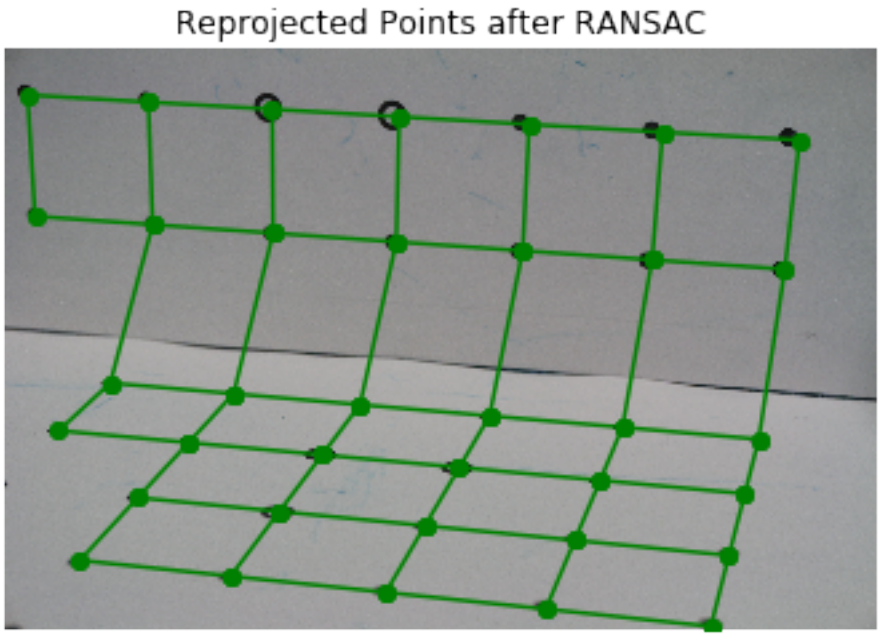
```
            K_best = X[0]
            print(X[0])
            print('Rotation Marix')
            print(X[1])
            print('Camera Centre')
            print(X[2])
```

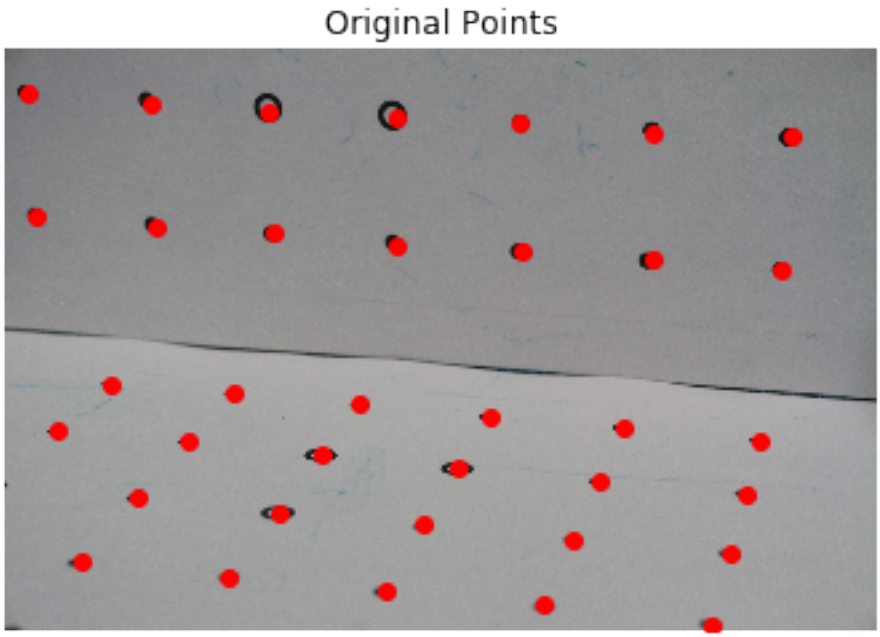
```
Camera Marix
[[ -1.26180700e+04   3.93200457e+02   2.84057436e+03]
 [ -0.00000000e+00  -1.23055399e+04   3.15771105e+03]
 [ -0.00000000e+00  -0.00000000e+00   1.00000000e+00]]
Rotation Marix
[[-0.97704029  0.00638989 -0.21295878]
 [-0.09979555 -0.89684288  0.43094512]
 [-0.18823688  0.44230308  0.87688931]]
Camera Centre
[ -25.78617935  204.713942    525.79695508]
```

```
In [515]: n = img_points.shape[0]
          W = np.zeros((n,4))
          W[:,0:3] = world_pts[:,0:3]
          W[:,3] = 1
          pp = np.zeros((n,3))
          for i in range(n):
              pp[i,:] = np.matmul(P,np.transpose(W[i,:]))
              pp[i,:] =  pp[i,:]/pp[i,2]

          ppp = []
          for aa in pp:
              ppp.append([int(aa[0]),int(aa[1])])
```

```
In [516]: reim = I.copy()
          plt.imshow(reim)
          plt.title('Reprojected Points after RANSAC')
          pts = [7,14,20,26,31,36]
          plt.axis('off')

          q = 0
          for i in range(35):
              if i == pts[q]-1:
                  q+=1
                  continue
              plt.plot([ppp[i][0],ppp[i+1][0]],[ppp[i][1],ppp[i+1][1]],'go-')
          idx = [7,6,6,5,5,4]
          num = [2,4,6,6,6,6,6]

          for i in range(7):
```

```
    prev = i
    for j in range(num[i]-1):
        nxt = prev + idx[j]
        plt.plot([ppp[prev][0],ppp[nxt][0]],[ppp[prev][1],ppp[nxt][1]],'go-')
        prev = nxt

plt.show()

reim1 = I.copy()
plt.imshow(reim1)
plt.title('Original Points')
plt.axis('off')

plt.plot(img_points[:,0],img_points[:,1],'ro')
plt.show()
```





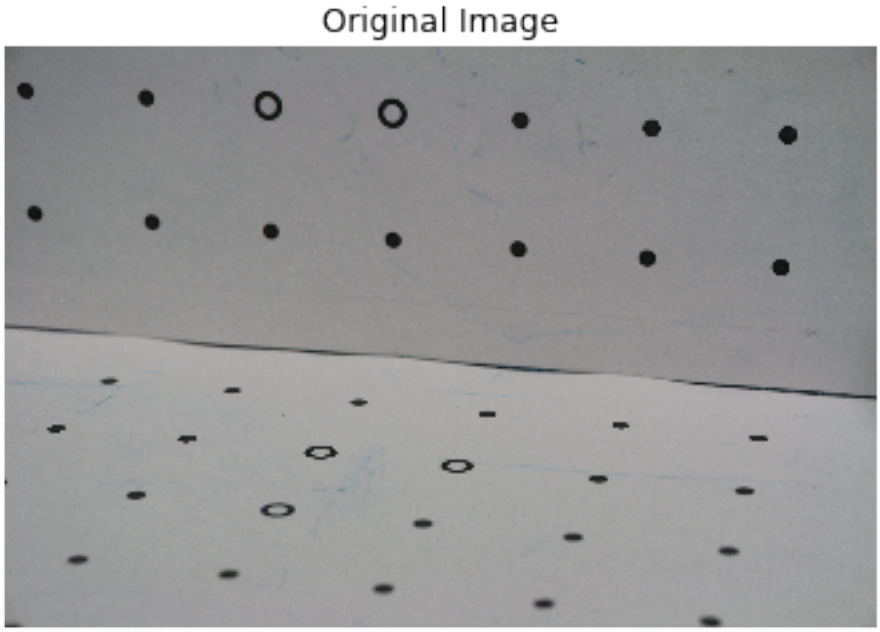
1.10 Radial Distortion Parameter Calculation

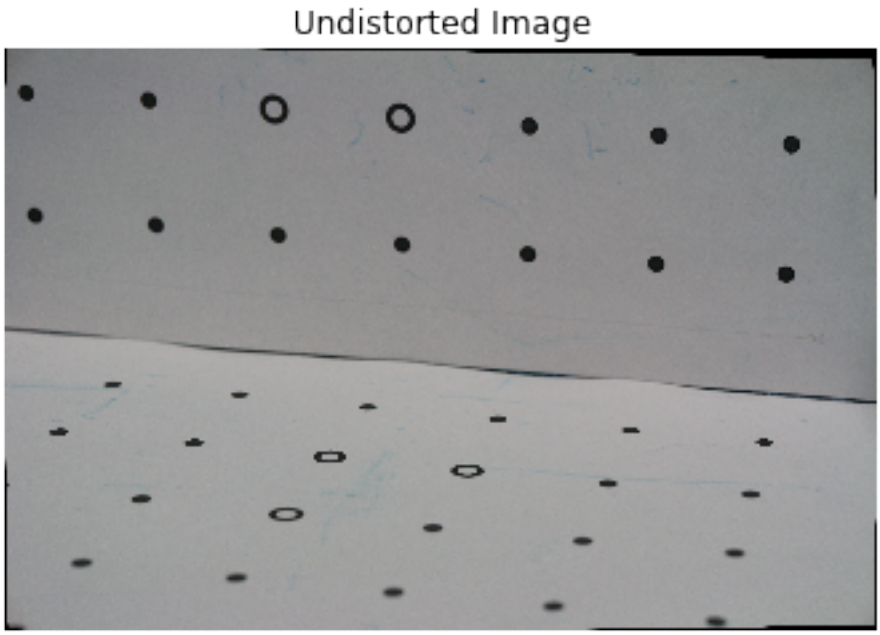
Here as can be seen in results below difference because of radial distortion is very less and hence the results of DLT and RANSAC will be almost similar to above obtained results

```
In [517]: I = cv2.imread('data/IMG_5455.JPG')
          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 = img_points[:,: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

          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.imshow(I)
          plt.title('Original Image')
```

```
plt.axis('off')
plt.show()
plt.imshow(I_undistort)
plt.axis('off')
plt.title('Undistorted Image')
plt.show()
```

```
Distortion Parameters are:
[[  6.90749628e-01  -7.92335856e+00   1.20551300e-02   6.50108294e-02
    4.30054841e+01]]
```

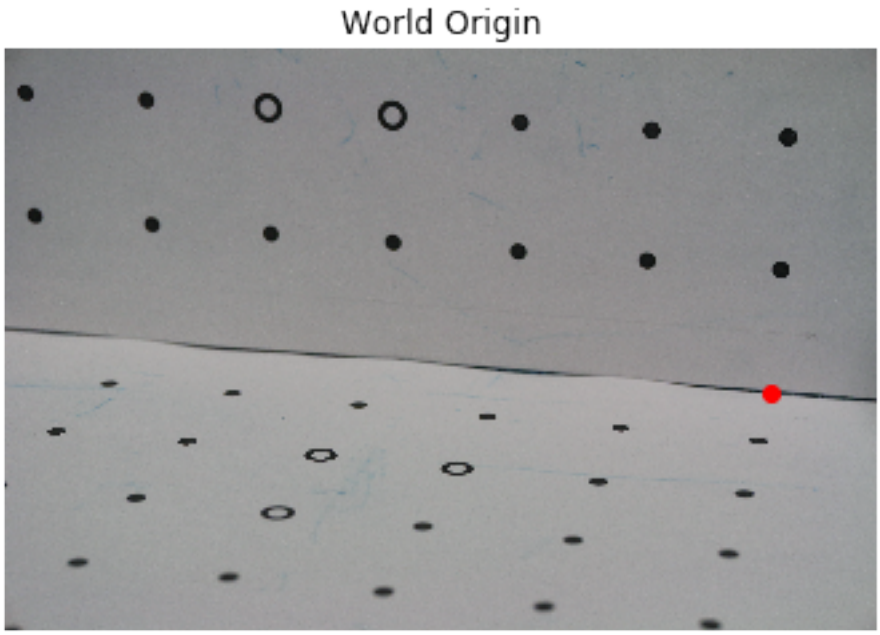




1.11 Image of world origin

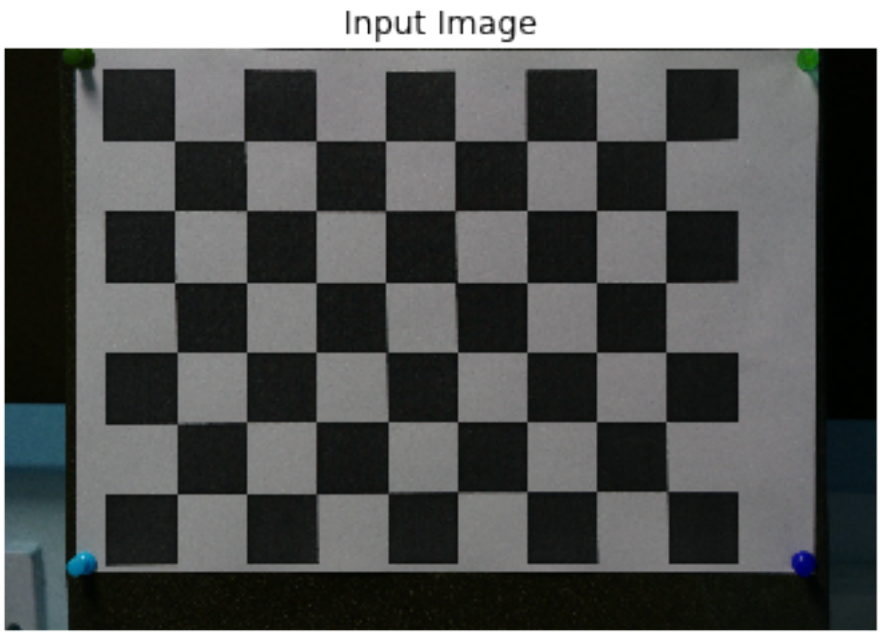
Observation: The point obtained using projection matrix is in accordance with origin specified in measurerments.png.

```
In [518]: AA = np.matmul(P,np.array([0,0,0,1]))
          plt.imshow(I)
          plt.title('World Origin')
          plt.axis('off')
          plt.plot([AA[0]],[AA[1]],'ro')
          plt.show()
```



1.12 Calibration using Zhangs Method

```
In [519]: im_left = cv2.imread('data/IMG_5456.JPG')
          plt.imshow(im_left)
          plt.title('Input Image')
          plt.axis('off')

          ret, corners = cv2.findChessboardCorners(im_left, (8,6))
          corners=corners.reshape(-1,2)
          x,y=np.meshgrid(range(8),range(6))
```

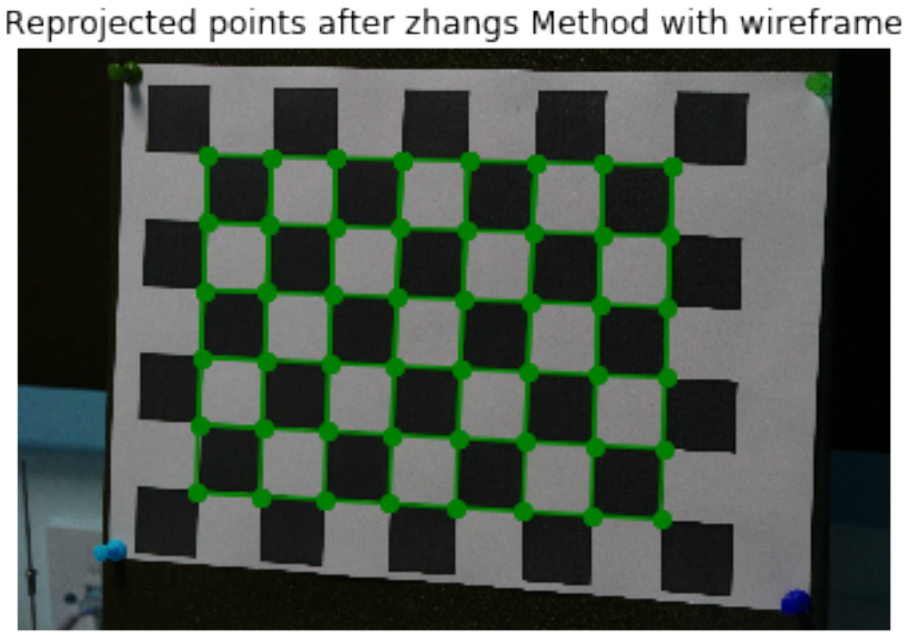


In [520]:
```
world_points=np.hstack((x.reshape(48,1),y.reshape(48,1),np.zeros((48,1)))).astype(np.f
_3d_points=[]
_2d_points=[]
for path in range(5456,5470):
    im=cv2.imread('data/IMG_' + str(path) + '.JPG')
    ret, corners = cv2.findChessboardCorners(im, (8,6))

    if ret: #add points only if checkerboard was correctly detected:
        _2d_points.append(corners) #append current 2D points
        _3d_points.append(world_points) #3D points are always the same


ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(_3d_points, _2d_points, (im.shape[1
```

In [521]:
```
num = world_points.shape[0]
world = np.zeros((num,4))
world[:,:2] = world_points[:,:2]
world[:,2] = 0
world[:,3] = 1
I = im.copy()
plt.imshow(cv2.imread('data/IMG_5457.JPG'))
plt.axis('off')
plt.title('Reprojected points after zhangs Method with wireframe')
rotation_mat = np.zeros(shape=(3, 3))
R = cv2.Rodrigues(rvecs[1], rotation_mat)[0]
```

```
P = np.column_stack((np.matmul(mtx,R), np.matmul(mtx, tvecs[1])))
P = P/P[2,3]
print('Projection Matrix')
print(P)
print('Camera Matrix')
print(KRT(P)[0])
projected_point = np.zeros((48, 3))

for i in range(num):
    projected_point[i,:] = np.matmul(P, world[i,:])
    projected_point[i,:] = projected_point[i,:] / projected_point[i,2]
p1 = projected_point[:,0]
p2 = projected_point[:,1]
ch = 7
for i in range(47):
    if i == ch:
        ch+=8
        i += 1
    plt.plot([p1[i],p1[i+1]],[p2[i],p2[i+1]],'go-')

for i in range(8):
    for j in range(5):
        plt.plot([p1[i],p1[i+(j+1)*8]],[p2[i],p2[i+(j+1)*8]],'go-')
```

```
Projection Matrix
[[  3.88386571e+02  -1.56645201e+01   1.95432985e+02   1.19140661e+03]
 [  4.10025718e+00   4.22348961e+02   5.03737446e+01   6.67251121e+02]
 [ -6.89177375e-03  -1.50393622e-04   3.02806359e-02   1.00000000e+00]]
Camera Matrix
[[ -1.35997371e+04   7.79358105e-14   3.36312443e+03]
 [  0.00000000e+00  -1.36159997e+04   1.48643588e+03]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00]]
```



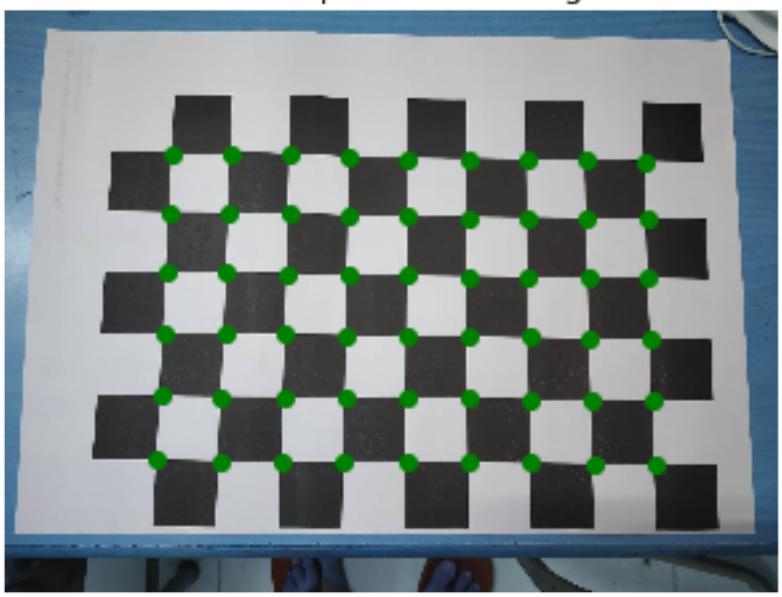
Observation: Focal length obtained in camera matrix along both the axes is similar for both of the methods. Though the principle point and scalling differs for both

2 Manual Images

2.1 Zhang Method

```
In [522]: im_left = cv2.imread('data1/im1.jpg')
          plt.imshow(im_left)
          plt.title('First Sample of data images')
          plt.axis('off')
          ret, corners = cv2.findChessboardCorners(im_left, (9,6))
          plt.plot(corners[:,0,0],corners[:,0,1],'go')
          corners=corners.reshape(-1,2)
          x,y=np.meshgrid(range(9),range(6))
          world_points=np.hstack((x.reshape(54,1),y.reshape(54,1),np.zeros((54,1)))).astype(np.f
          # print(world_points)
```

First Sample of data images

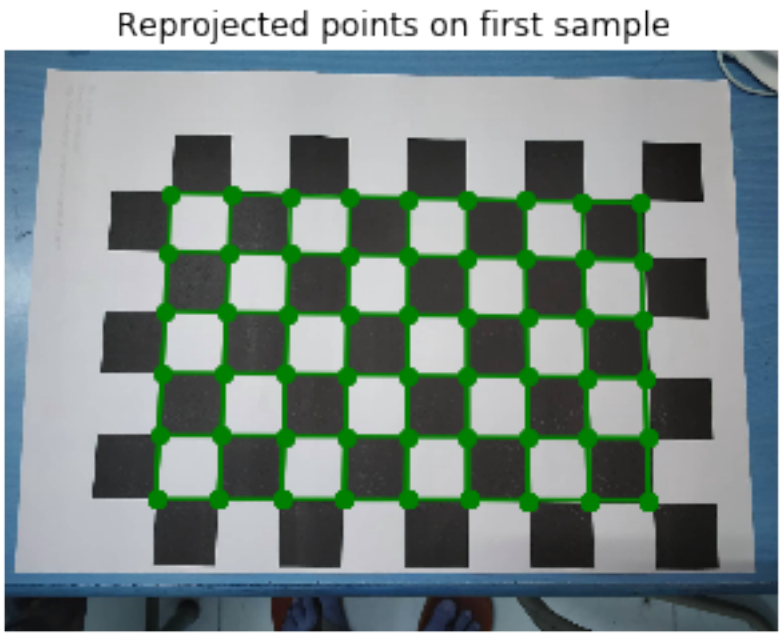


In [523]:
```
_3d_points=[]
_2d_points=[]
for path in range(1,18):
    im=cv2.imread('data1/im' + str(path) + '.jpg')
    ret, corners = cv2.findChessboardCorners(im, (9,6))
    if ret: #add points only if checkerboard was correctly detected:
        _2d_points.append(corners) #append current 2D points
        _3d_points.append(world_points) #3D points are always the same

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(_3d_points, _2d_points, (im.shape[1
```

In [524]:
```
num = world_points.shape[0]
world = np.zeros((num,4))
world[:,:2] = world_points[:,:2]
world[:,2] = 0
world[:,3] = 1
plt.imshow(cv2.imread('data1/im1.jpg'))
plt.title('Reprojected points on first sample')
plt.axis('off')

rotation_mat = np.zeros((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]
projected_point = np.zeros((54, 3))
```

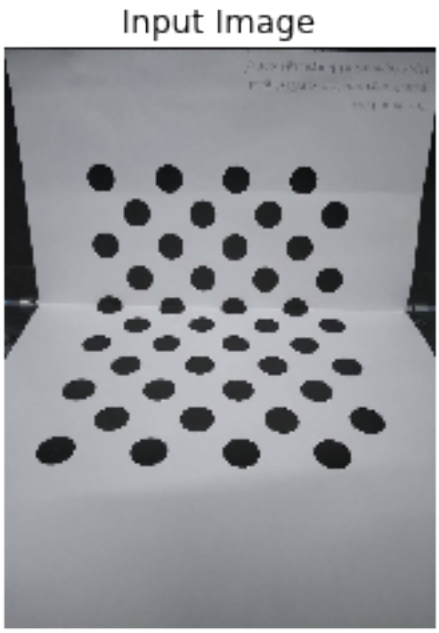
```
print('Projection Matrix')
print(P)
for i in range(num):
    projected_point[i,:] = np.matmul(P, world[i,:])
    projected_point[i,:] = projected_point[i,:] / projected_point[i,2]

p1 = projected_point[:,0]
p2 = projected_point[:,1]
ch = 8
for i in range(53):
    if i == ch:
        ch+=9
        i += 1
    plt.plot([p1[i],p1[i+1]],[p2[i],p2[i+1]],'go-')

for i in range(9):
    for j in range(5):
        plt.plot([p1[i],p1[i+(j+1)*9]],[p2[i],p2[i+(j+1)*9]],'go-')
```

```
Projection Matrix
[[ -1.60835365e+02   1.22961640e+01   9.51673209e+01   1.66222804e+03]
 [ -4.84017163e+00  -1.49333754e+02   8.89551346e+01   1.16050426e+03]
 [ -3.82941174e-03   1.05703521e-02   9.79025558e-02   1.00000000e+00]]
```



2.2 Calibrating own camera using DLT

```
In [525]: I = cv2.imread('data/dlt.jpg')
          plt.imshow(I)
          plt.title('Input Image')
          plt.axis('off')
          plt.show()
```




```
In [526]: world_cords = [[6,3,0],[4,3,0],[2,3,0],[0,3,0],[6,1,0],[4,1,0],[2,1,0],[0,1,0],[6,0,1]
          img_cords = [[451,569],[685,580],[904,573],[1139,576],[487,799],[689,799],[901,796],[1
          world_cords = np.array(world_cords)
          img_cords = np.array(img_cords)
```

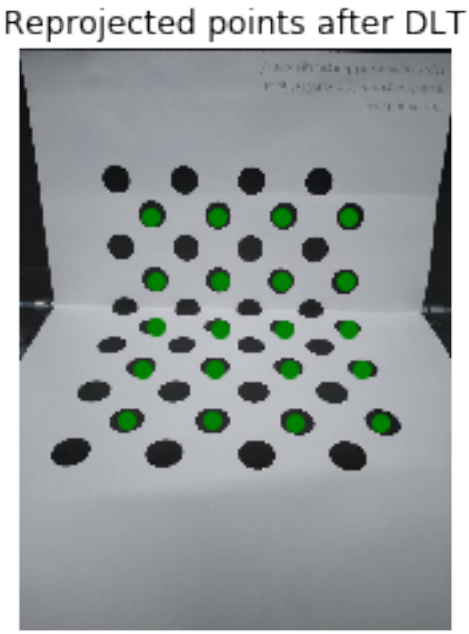
```
In [527]: P1 = DLT(world_cords,img_cords)
          print('Projection matrix')
          print(P1)
          krt = KRT(P1)
          print('Camera Matrix')
          print(krt[0])
          print('Rotation Matrix')
          print(krt[1])
          print('Camera Centre')
          print(krt[2])
```

```
Projection matrix
[[ -1.04363579e+02  -2.17234082e+01  -4.32482976e+01   1.11426450e+03]
```

```
 [  6.92901411e-01  -1.23734657e+02   4.65580663e+00   9.01250429e+02]
 [  1.01749029e-03  -2.55880192e-02  -5.59053582e-02   1.00000000e+00]]
Camera Matrix
[[ -1.70996611e+03  -2.98511982e+01   7.58354064e+02]
 [ -0.00000000e+00  -1.86119365e+03   7.68685103e+02]
 [ -0.00000000e+00  -0.00000000e+00   1.00000000e+00]]
Rotation Matrix
[[ -9.99862787e-01  -6.17747232e-03  -1.53702869e-02]
 [ -7.79643995e-04  -9.09287074e-01   4.16168728e-01]
 [ -1.65468740e-02   4.16123607e-01   9.09157492e-01]]
Camera Centre
[  3.09597399   7.84120429  14.35478215]
```

```
In [528]: n = img_cords.shape[0]
          W = np.zeros((n,4))
          W[:,0:3] = world_cords[:,0:3]
          W[:,3] = 1
          pp = np.zeros((n,3))
          for i in range(n):
              pp[i,:] = np.matmul(P1,np.transpose(W[i,:]))
              pp[i,:] =  pp[i,:]/pp[i,2]

          ppp = []
          for aa in pp:
              ppp.append([int(aa[0]),int(aa[1])])
          ppp = np.array(ppp)
```

```
In [529]: plt.imshow(I)
          plt.plot(ppp[:,0],ppp[:,1],'go')
          plt.title('Reprojected points after DLT')
          plt.axis('off')
          plt.show()
```



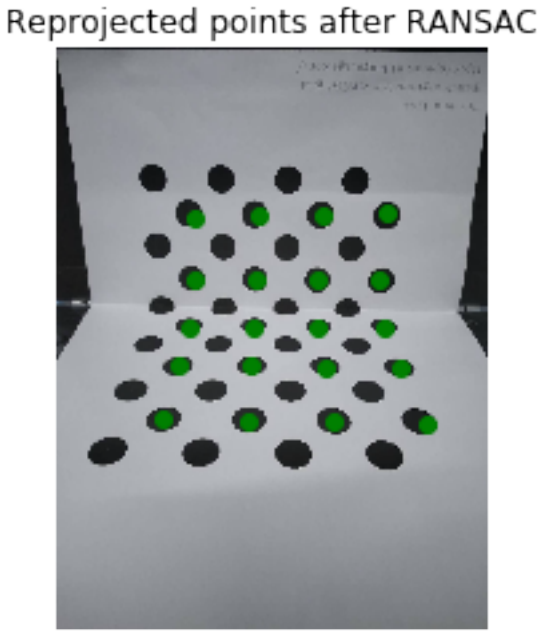

```
In [530]: P = RANSAC(img_cords,world_cords)
          print('Projection Matrix')
          print(P)
          X = KRT(P)
```

```
Max Inlier Update to ----> 1
Max Inlier Update to ----> 2


/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: divide by zero
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: invalid value e
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:17: RuntimeWarning: divide by zero
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:17: RuntimeWarning: invalid value e


Max Inlier Update to ----> 3
Projection Matrix
[[ -1.00870941e+02  -8.03592212e+00  -4.64021442e+01   1.10740207e+03]
 [  6.66199336e+00  -1.21890399e+02  -3.36727894e+00   9.02723812e+02]
 [  7.10175544e-03  -1.84211451e-02  -6.31318529e-02   1.00000000e+00]]
```

```
In [531]: n = img_cords.shape[0]
          W = np.zeros((n,4))
```

```
W[:,0:3] = world_cords[:,0:3]
W[:,3] = 1
pp = np.zeros((n,3))
for i in range(n):
    pp[i,:] = np.matmul(P,np.transpose(W[i,:]))
    pp[i,:] =  pp[i,:]/pp[i,2]

ppp = []
for aa in pp:
    ppp.append([int(aa[0]),int(aa[1])])
ppp = np.array(ppp)
```

In [532]:
```
plt.imshow(I)
plt.plot(ppp[:,0],ppp[:,1],'go')
plt.title('Reprojected points after RANSAC')
plt.axis('off')
plt.show()
```



3 Challenges Faced

- Because of radial distortion in some cases the reprojected points were not same as the actual points
- optimal solution of RANSAC gave around 2-4 inliers which is very less
- Manually finding coordinated for DLT was challenging job (As Harris COrner detector was not giving all points)

4 Learnings from Assignment

- Learned about various algorithm and their practical applications
- Learned about various flags and features of cameraCalibrate function which was used fior Zhangs Method and also for calculating radial distortion parameters