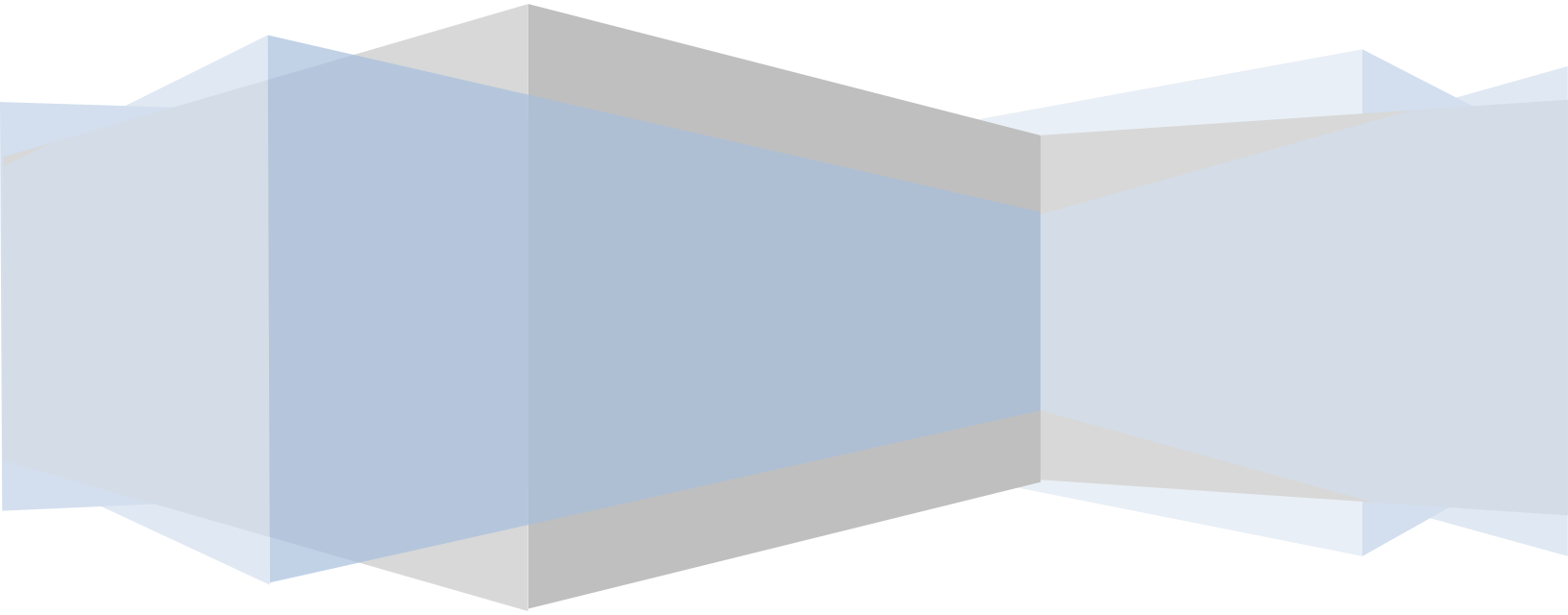
International Institute of Information Technology, Hyderabad

Camera Calibration

DLT, Zhang's method

Mohit Sharma

201505508



DLT (Direct Linear Transformation) based Calibration

Implementation steps:

1. Noted 6 world points and corresponding image points.
2. Created matrix A using these points and ran SVD on the same to find out projection matrix. Code for the same is attached at the end.

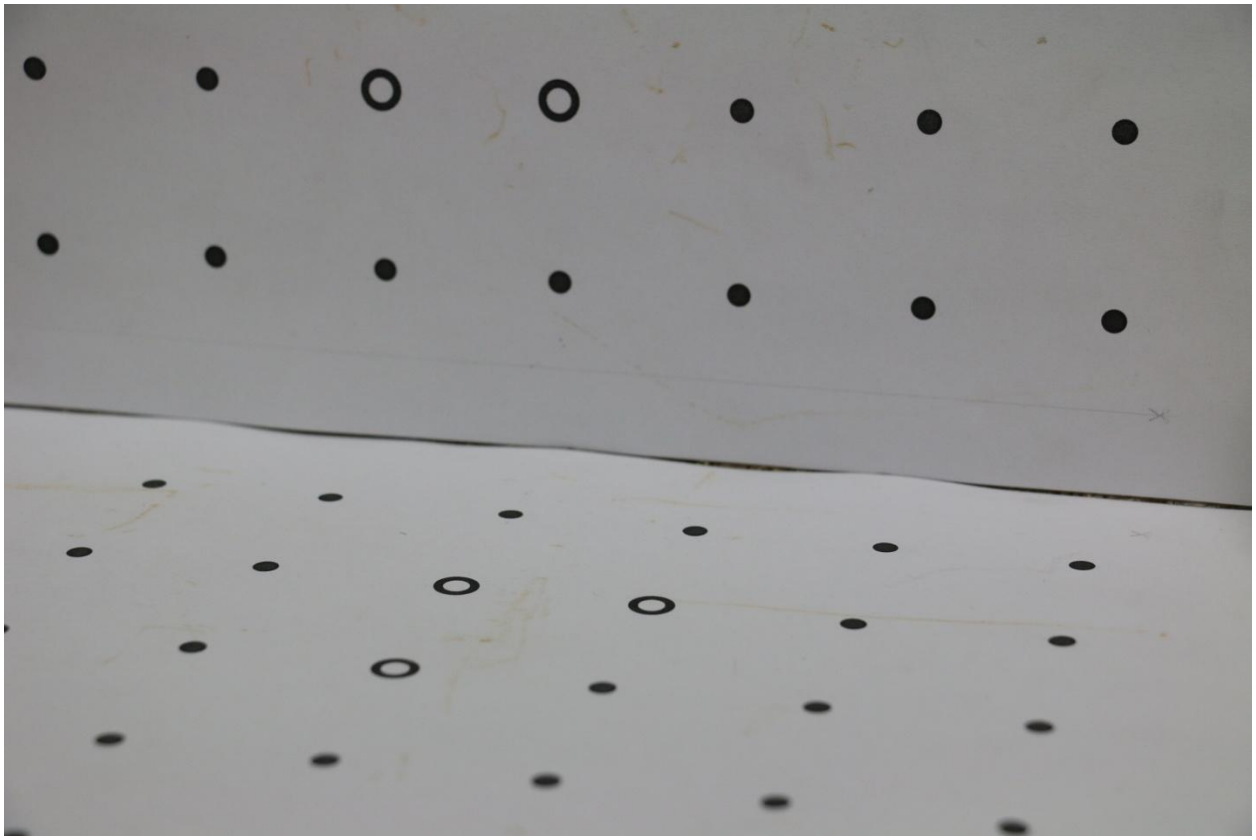


Image used for DLT

Results:

Projection matrix:

```
[[ -3.74192548e-03   -7.09352321e-04  -2.08639185e-03   9.11534249e-01]
 [ -1.19471698e-04   -4.38859509e-03   6.67848748e-04   4.11177210e-01]
 [  1.03536464e-07   -2.21616919e-07  -3.72293950e-07   1.90809706e-04]]
```

Camera's intrinsic parameters:

```
[[  4.16539126e-03    4.99336916e-05   1.22688234e-03]
 [  0.00000000e+00    4.14347407e-03   1.59739944e-03]
 [ -0.00000000e+00    0.00000000e+00   4.45462281e-07]]
```

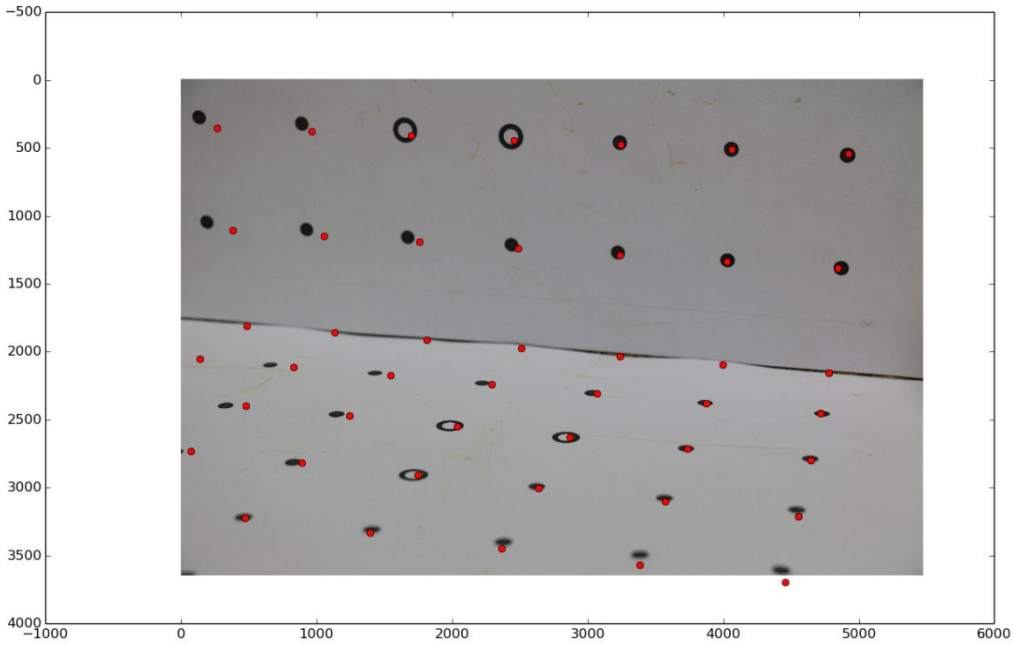
Camera's extrinsic parameters:

```
[[ 0.96537612         0.01336473          0.26051935]
 [-0.11843851         -0.86736175         0.48337969]
 [ 0.23242476         -0.49749873         -0.83574742]]
```

Observation for the 6 points:

S. No.	World Point	Image Point	Calculated point
1	144, 0, 144	475, 3222	475.03, 3222.32
2	36, 0, 36	3874, 2379	3874, 2378.99
3	36, 72, 0	4064, 509	4064, 508.99
4	72, 72, 0	3242, 475	3241.99, 475
5	72, 0, 108	2637, 3004	2636.93, 3003.37
6	0, 0, 72	4642, 2800	4642.03, 2800.30

Wireframe image for the DLT:



Please note that only points have been plotted for clarity

Ransac based calibration:

Implementation steps:
Image coordinates for 18 points are taken and followed these steps to calibrate:

1. Randomly select 6 points out of 18.
2. Find P matrix for above selected points using DLT.
3. Using this P matrix, Find out image coordinates (xi, yi) for all the 18 points.
4. Calculate the error in terms of Euclidean distance from calculated values (xi, yi) to actual values.
5. If this error is lesser than the threshold (taken as 1) then return P matrix as the final result.
6. Repeat above 5 steps until we reach threshold.

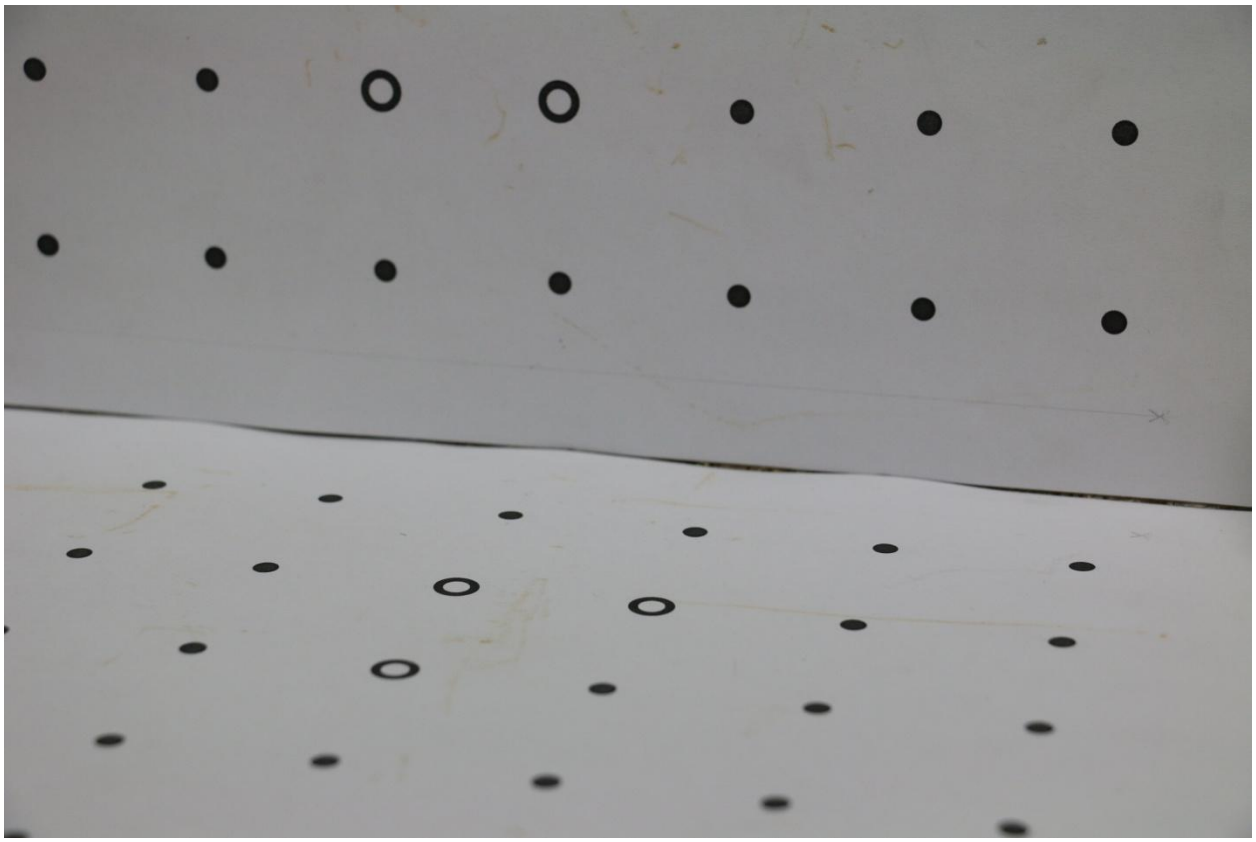


Image used for Ransac

Results:

Projection matrix:

```
[[  3.96173182e-03   3.11538970e-04   1.86238666e-03  -9.13153168e-01]
 [  1.87708116e-04   4.24084506e-03  -8.65777998e-04  -4.07569902e-01]
 [ -7.63671423e-08   1.19359844e-07   3.05929301e-07  -1.89856060e-04]]
```

Camera intrinsic parameters:

```
[[  4.29469467e-03   3.63134684e-05  -9.02850295e-04]
 [  0.00000000e+00   4.27975669e-03  -6.73242439e-04]
 [ -0.00000000e+00   0.00000000e+00  -3.37151969e-07]]
```

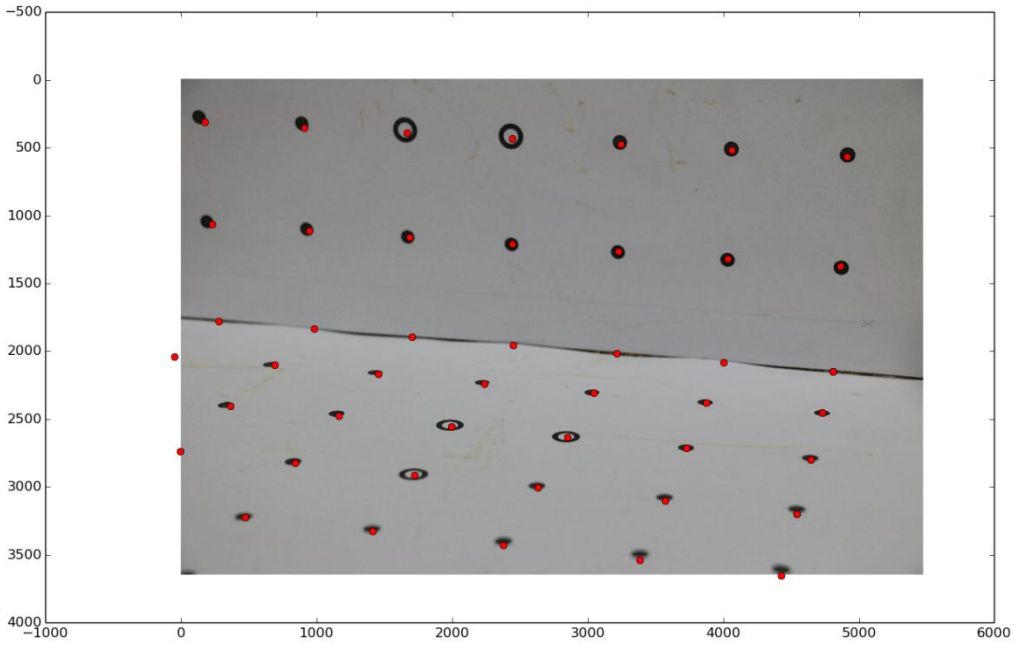
Camera extrinsic parameters:

```
[[ 0.9707605   0.00602357  0.23997451]
 [-0.07949096 -0.93521699  0.34503677]
 [ 0.22650659 -0.35402387 -0.9073929 ]]
```

Observation for the 6 points:

S. No.	World Point	Image Point	Calculated point
1	0, 0, 36	4731, 2453	4731.01, 2453.20
2	0, 36, 0	4873, 1407	4860.64, 1373.68
3	0, 72, 0	4914, 564	4914, 563.98
4	36, 0, 0	4010, 2073	4000.56, 2081.00
5	0, 0, 0	4826, 2141	4809.71, 2146.73
6	0, 0, 72	4642, 2800	4641.99, 2799.90

Wireframe image for the Ransac:

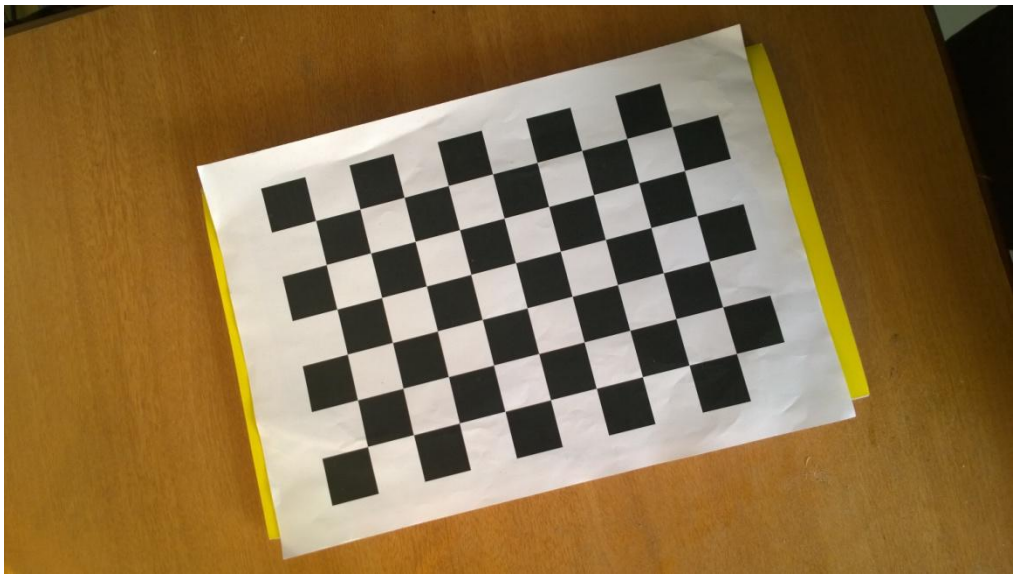


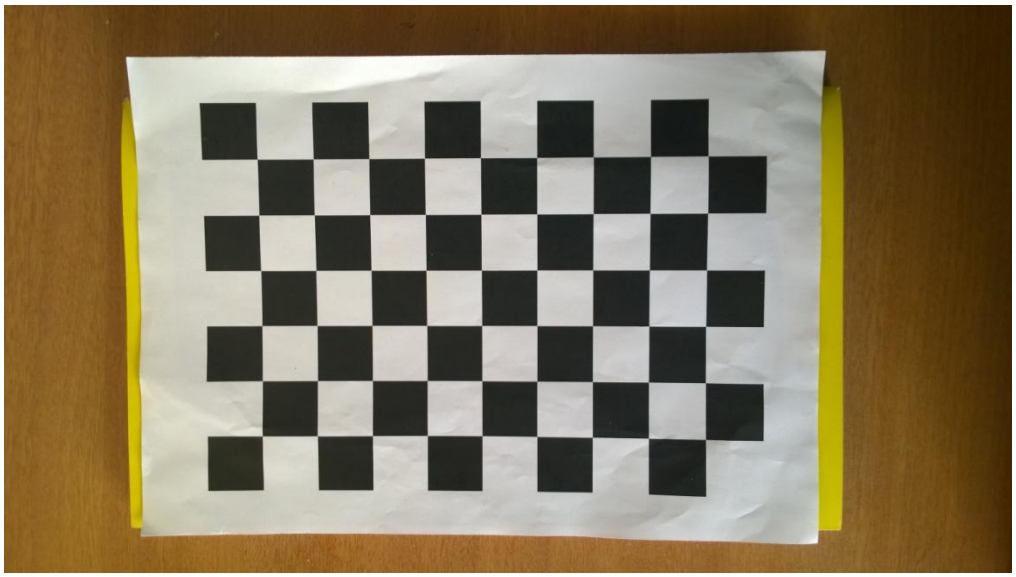
Please note that only points have been plotted for clarity

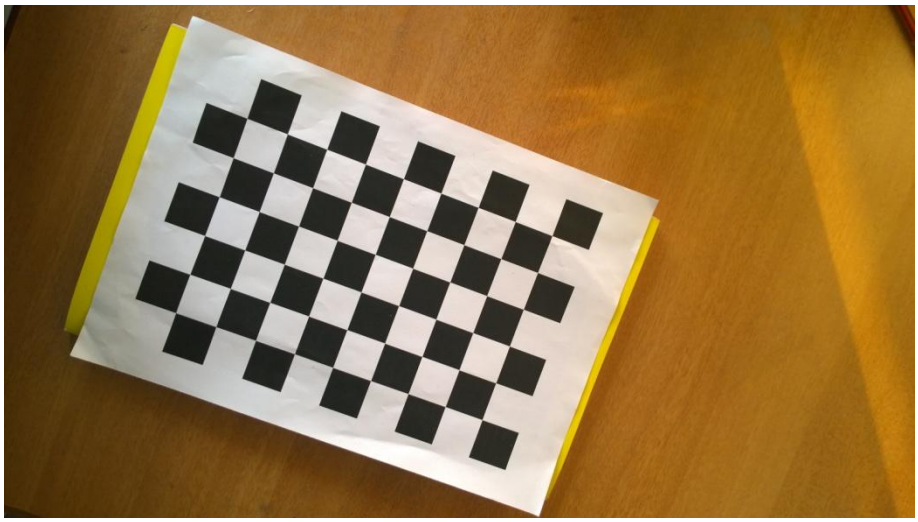
As we can see in the wireframe that error in calculating the image points is very low as compared to DLT. So projection matrix calculated using Ransac is better than that of DLT

Zhang's to calibrate mobile phone camera:

Captured following planar images of checker board and gave as input to opencv implementation of zhang's calibration method:







Camera's intrinsic parameters reported by zhang's method:

2.3429210562070466e+03	0.	1.5355000000000000e+03
0.	2.3429210562070466e+03	8.6350000000000000e+02
0.	0.	1.

DLT to calibrate mobile phone camera:

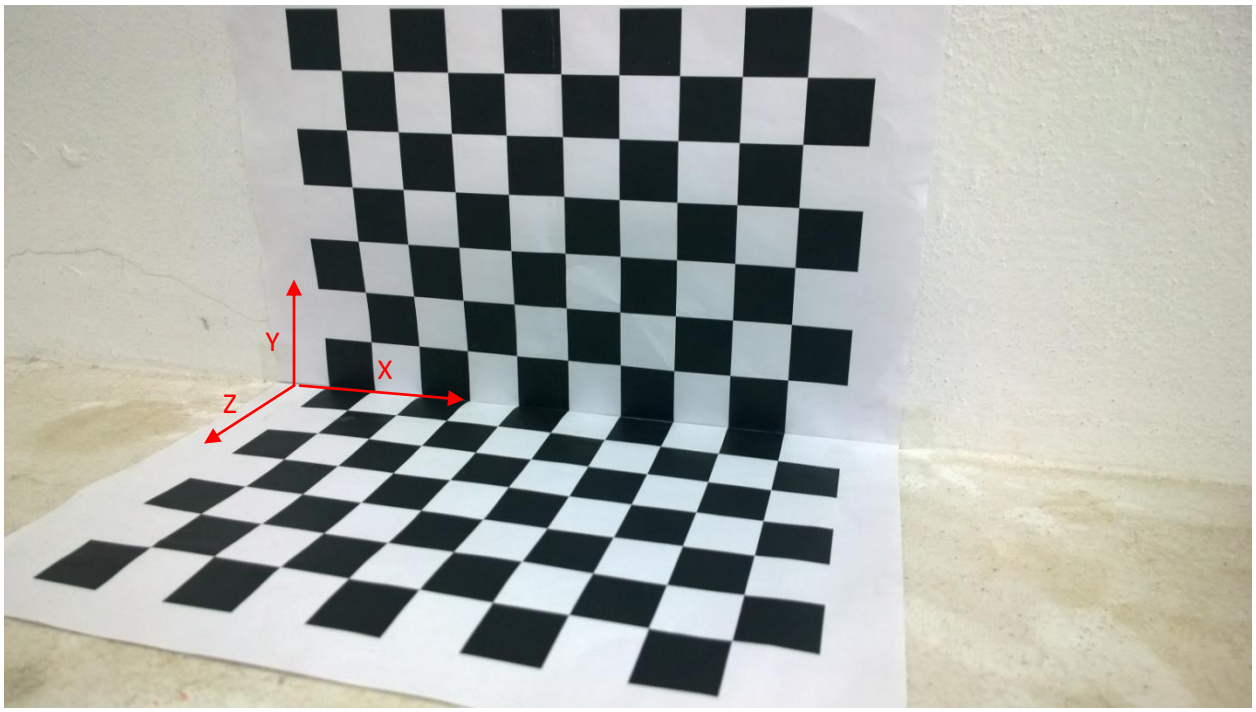


Image used for DLT

Same implementation of DLT as mentioned in 1st part is used

Camera's intrinsic parameters reported by DLT method:

```
[[ 3.53671254e+03    0.0000012e-00    1.86234251e+03 ]
 [  0.00000000e+00    2.61537820e+03    8.47719485e+02]
 [  0.00000000e+00    0.00000000e+00    1]]
```

Please note above matrix is scaled to have a 1 in bottom right cell.

Codes

```
# program to implement DLT method to calibrate the camera

from numpy import *
import numpy as np
from scipy import linalg

A =[]
x = [[733, 989], [792,821], [775, 698], [912, 950], [803, 939], [651,
1045]]          # checkerboard
X = [[0, 0, 25], [0,25,0], [0,50,0], [25,0,0], [0,0,0], [0,0,50]]

x = [[475, 3222], [3874, 2379], [4064, 509], [3242, 475], [2637,
3004], [4642, 2800]]  # provided image
X = [[144, 0, 144], [36, 0, 36], [36, 72, 0], [72, 72, 0], [72, 0,
108], [0, 0, 72]]

def addPointToA(x,X):                                          # Add
image point x and world point X to A
     l = [X[0],X[1],X[2],1,0,0,0,0,-1*x[0]*X[0], -1 * x[0]*X[1], -
1*x[0]*X[2], -1*x[0]]
     A.append(l)
     l = [0,0,0,0,X[0],X[1],X[2],1,-1*x[1]*X[0], -1 * x[1]*X[1], -
1*x[1]*X[2], -1*x[1]]
     A.append(l)

def prepareA():
     #prepare matrix A for selected points
     for i in range(len(x)):
          addPointToA(x[i], X[i])

prepareA()

print A
U,s,V = linalg.svd(A)
P = V[-1].reshape((3,4))
M =[]
for i in P:
     M.append(i[0:3])

r, q = linalg.rq(M)
print 'Projection matrix is:'
print P
print 'camera intrinsic parameters are:'
print r
print
print 'camera extrinsic parameters are:'
print q
```

```
print 'Image points calculated using this camera matrix:'
for i in range(len(X)):
      y = np.dot(P, X[i] + [1])
      print 'World point: %s' % X[i]
      print 'Actual Image point: %s' % x[i]
      l = [y[0]/y[2], y[1]/y[2]]
      print 'Calculated point: %s' % l
      print
```

 # Program to implement ransac to calibrate camera

```
from random import randint
from numpy import *
import numpy as np
from scipy import linalg
import math

A =[]

# 18 image points and world points
x = [[4731, 2453], [4873,1407], [4914, 564], [4010, 2073], [4826,
2141], [4642, 2800], [3874, 2379], [3038,2304], [4037, 1325],
[4064,509], [3242, 475], [2440,1216], [3731,2712],[1155,2460],
[2637,3004], [836,2807], [475,3222], [897,333]]
X = [[0, 0, 36], [0,36,0], [0,72,0], [36,0,0], [0,0,0], [0,0,72],
[36,0, 36], [72,0,36], [36,36,0], [36,72,0], [72,72,0],[108,36,0],
[36,0,72], [140,0,72],[72,0,108],[144,0,108], [144,0,144], [180,72,0]]

def addPointToA(x,X):
      l = [X[0],X[1],X[2],1,0,0,0,0,-1*x[0]*X[0], -1 * x[0]*X[1], -
1*x[0]*X[2], -1*x[0]]
      A.append(l)
      l = [0,0,0,0,X[0],X[1],X[2],1,-1*x[1]*X[0], -1 * x[1]*X[1], -
1*x[1]*X[2], -1*x[1]]
      A.append(l)

def prepareA():
      d ={}
      for i in range(6):
            t = randint(0,17)
            while t in d:
                  t = randint(0,17)
            d[t] = 1
            addPointToA(x[t], X[t])

P = []

threshold = 1
def getTotalError():                  # return error in calculated
points using current projection matrix
      dist = 0
      for i in range(0,12,2):
            WP = [A[i][0], A[i][1], A[i][2]]
            y = np.dot(P, WP + [1])
            dist += (-1 * A[i][11] - y[0]/y[2]) * (-1 * A[i][11] -
y[0]/y[2]) + (-1 * A[i+1][11] - y[1]/y[2]) * (-1 * A[i+1][11] -
y[1]/y[2])
      return math.sqrt(dist)

while 1:
```

```
      A = []
      prepareA()
      U,s,V = linalg.svd(A)
      P = V[-1].reshape((3,4))
      if(getTotalError() <= threshold):
            break;


M =[]
for i in P:
      M.append(i[0:3])

r, q = linalg.rq(M)

print 'Projection matrix is:'
print P
print 'camera intrinsic parameters are:'
print r
print
print 'camera extrinsic parameters are:'
print q

print 'Image points calculated using this camera matrix:'
for i in range(6):
      y = np.dot(P, X[i] + [1])
      print 'World point: %s' % X[i]
      print 'Actual Image point: %s' % x[i]
      l = [y[0]/y[2], y[1]/y[2]]
      print 'Calculated point: %s' % l
      print
```

#program to plot wireframe points for DLT and ransac, please note only points are plotted for clarity, lines joining them aren't drawn

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

img=mpimg.imread('IMG_5455.JPG')
imgplot = plt.imshow(img)

P = [[ -3.74192548e-03,  -7.09352321e-04,  -2.08639185e-03,
9.11534249e-01],                 #DLT
 [ -1.19471698e-04,  -4.38859509e-03,   6.67848748e-04,   4.11177210e-
01],
 [  1.03536464e-07,  -2.21616919e-07,  -3.72293950e-07,   1.90809706e-
04]]

P = [[  3.96173182e-03,   3.11538970e-04,   1.86238666e-03,  -
9.13153168e-01],                 #Ransac
 [  1.87708116e-04,   4.24084506e-03,  -8.65777998e-04,  -4.07569902e-
01],
 [ -7.63671423e-08,   1.19359844e-07,   3.05929301e-07,  -1.89856060e-
04]]

#all world points for provided image
X = [[[0, 72, 0], [36, 72, 0], [72, 72, 0], [108, 72, 0], [144, 72,
0], [180, 72, 0], [216, 72, 0]],
[[0, 36, 0], [36, 36, 0], [72, 36, 0], [108, 36, 0], [144, 36, 0],
[180, 36, 0], [216, 36, 0]],
[[0, 0, 0], [36, 0, 0], [72, 0, 0], [108, 0, 0], [144, 0, 0], [180, 0,
0], [216, 0, 0]],
[[0, 0, 36], [36, 0, 36], [72, 0, 36], [108, 0, 36], [144, 0, 36],
[180, 0, 36], [216, 0, 36]],
[[0, 0, 72], [36, 0, 72], [72, 0, 72], [108, 0, 72], [144, 0, 72],
[180, 0, 72]],
[[0, 0, 108], [36, 0, 108], [72, 0, 108], [108, 0, 108], [144, 0,
108], [180, 0, 108]],
[[0, 0, 144], [36, 0, 144], [72, 0, 144], [108, 0, 144], [144, 0,
144]],
[[0, 72, 0], [0, 36, 0], [0 ,0 ,0], [0 ,0, 36], [0 ,0, 72], [0 ,0,
108]]]


for i in X:
     for j in i:
          y = np.dot(P, j + [1])
          plt.plot(y[0]/y[2], y[1]/y[2], 'ro');

plt.show()
```