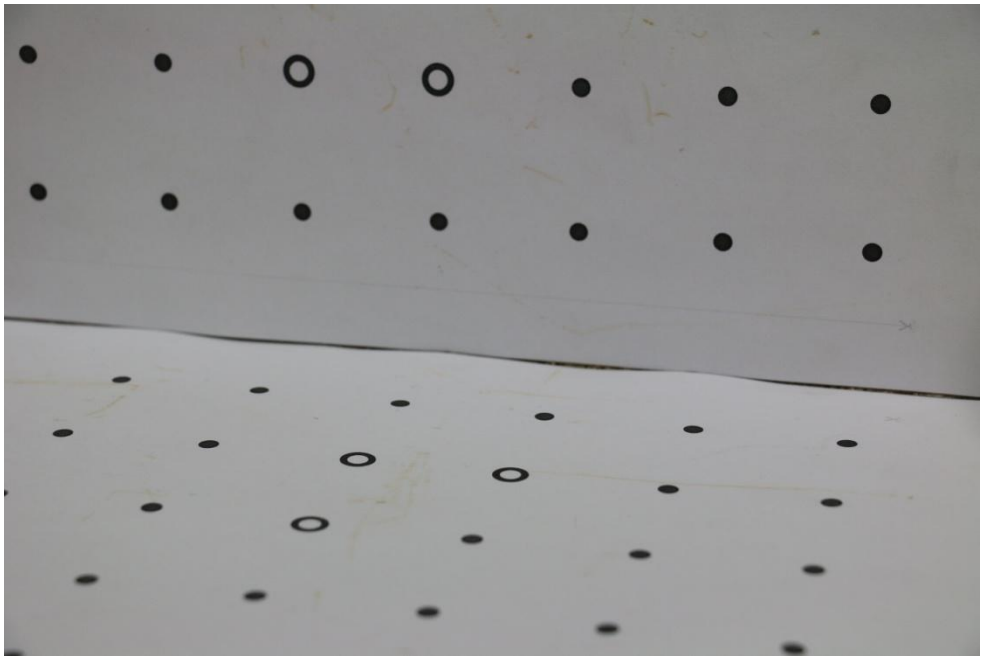
Assignment 1 (Camera calibration) Report

DLT implementation using RANSAC strategy

I used the following image to implement DLT.



and noted image coordinates for 20 points. Then followed following steps to calibrated:

1. Randomly select 6 points out of 20.
2. Find P matrix using above selected points using DLT.
3. Using this P matrix, Find out image coordinates (xi, yi) for all the 20 points.
4. Calculate the error in terms of Euclidean distance from calculated values to actual values.
5. If this error is lesser than previously recorded error till now then select this P matrix as the final result.
6. Repeat above 5 steps for 20 iterations.

Implementation for DLT is done in python:

Sample of 6 randomly chosen points during one iteration:

Image point	*World point*
A (4731, 2453)	(0, 0, 36)
B (4873, 1407)	(0, 36, 0)
C (4914, 564)	(0, 72, 0)
D (4010, 2073)	(36, 0, 0)
E (4826, 2141)	(0, 0, 0)
F (4642, 2800)	(0, 0, 72)

“A” matrix generated for the above sample:

```
A = [[0, 0, 36, 1, 0, 0, 0, 0, 0, 0, -170316, -4731],
     [0, 0, 0, 0, 0, 0, 36, 1, 0, 0, -88308, -2453],
     [0, 36, 0, 1, 0, 0, 0, 0, 0, -175428, 0, -4873],
     [0, 0, 0, 0, 0, 36, 0, 1, 0, -50652, 0, -1407],
     [0, 72, 0, 1, 0, 0, 0, 0, 0, -353808, 0, -4914],
     [0, 0, 0, 0, 0, 72, 0, 1, 0, -40608, 0, -564],
     [36, 0, 0, 1, 0, 0, 0, 0, -144360, 0, 0, -4010],
     [0, 0, 0, 0, 36, 0, 0, 1, -74628, 0, 0, -2073],
     [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -4826],
     [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, -2141],
     [0, 0, 72, 1, 0, 0, 0, 0, 0, 0, -334224, -4642],
     [0, 0, 0, 0, 0, 0, 72, 1, 0, 0, -201600, -2800]]
```

Homography with minimum error calculated after 20 iterations:

```
U,s,V = linalg.svd(A)
print V[-1]
```

```
[[-0.88832046543897425, -2.9969624329960476e-12, -2.7001213764865639e-12,
1.4737601755948902e-09],
[-0.45922402115784589, -7.0692128698022749e-12, 1.6686874080157072e-12,
6.5386712709167657e-10],
[-0.00022152630060649853, -6.6960326172704754e-16, -4.163336342344337e-16,
3.0544172232975248e-13]]
```

RQ decomposition of above matrix to generate intrinsic and extrinsic parameters:

```
x = V[-1].reshape((3,4))
M =[]
for i in x:
    M.append(i[0:3])
r, q = linalg.rq(M)
print r
print
print q
```

Matrix for intrinsic parameters:

```
[[  1.06841647e-12  -1.34704366e-13   8.88320465e-01]
 [  0.00000000e+00   1.09561637e-12   4.59224021e-01]
 [  0.00000000e+00   0.00000000e+00   2.21526301e-04]]
```

Matrix for Extrinsic parameters:

```
[[  2.94686497e-12  -4.07044850e-01  -9.13408173e-01]
 [  1.99601446e-12  -9.13408173e-01   4.07044850e-01]
 [ -1.00000000e+00  -3.02268065e-12  -1.87938554e-12]]
```

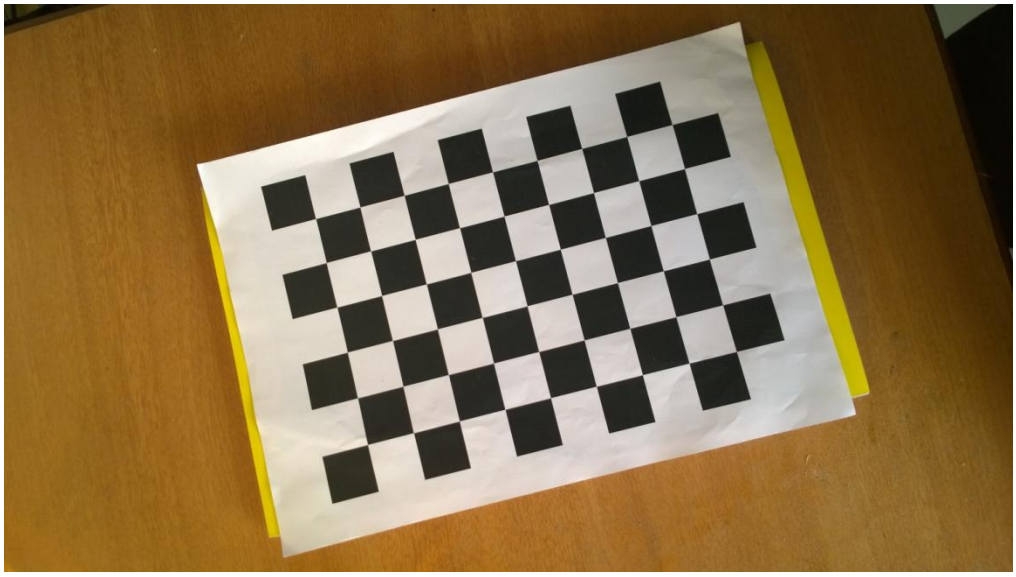
Comparison of actual image points and ones generated from calculated P matrix:

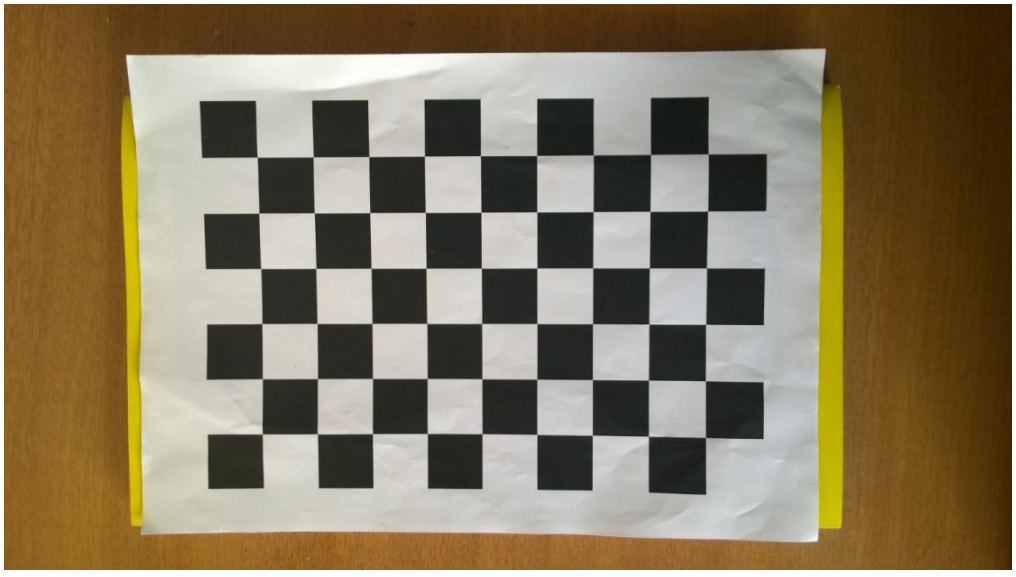
y = np.dot(x, [X,Y,Z,1])
print y[0]/y[2]
print y[1]/y[2]

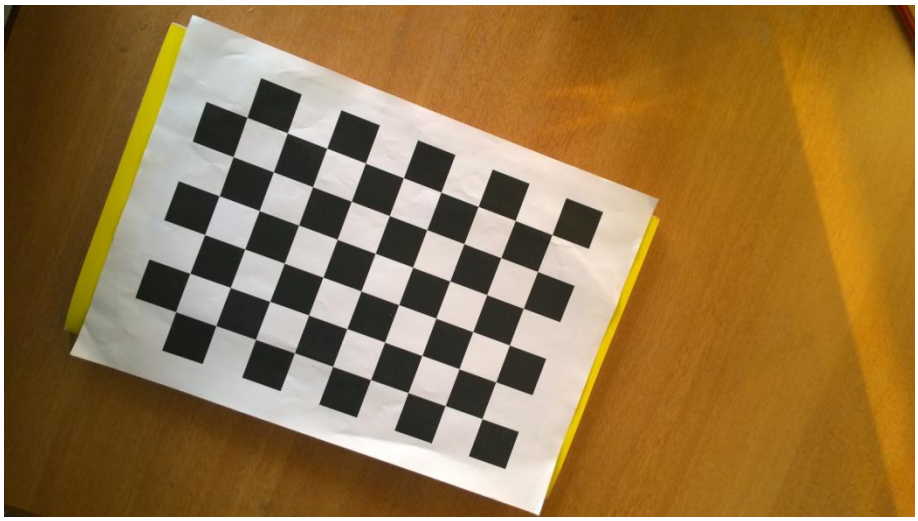
S. No.	World Point	Actual Image Point	Calculated Point
1.	0, 0, 0	4826, 2141	4825.01, 2140.7
2.	0, 0, 36	4731, 2453	4766.4, 2357.6
3.	0, 36, 0	4873, 1407	4845. 2, 1430.6
4.	0 , 72, 0	4914, 564	4868.01, 568.5
5	36, 0 , 0	4010, 2073	4009.9, 2073

Zhang's and DLT to calibrate mobile phone camera:

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



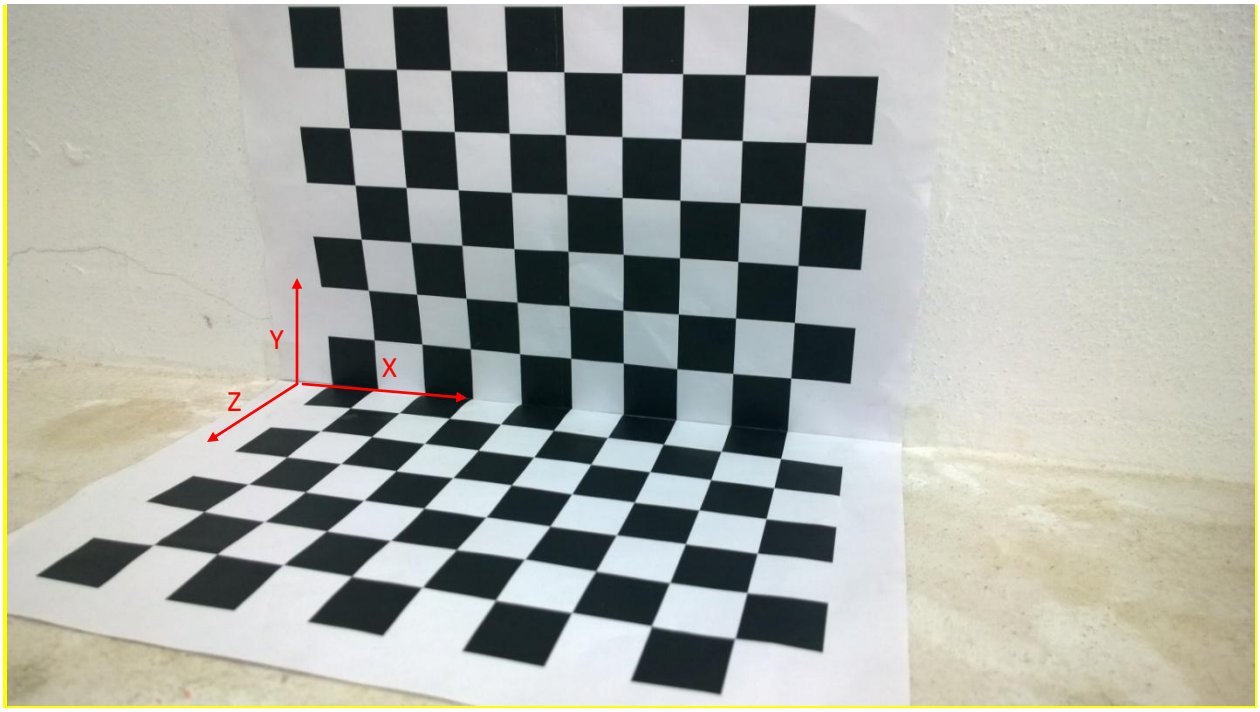




Camera parameters reported by zhang's method:

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

Camera parameters reported by DLT method:
Image used for implementing DLT: World coordinates chosen are marked in red



Same implementation of DLT as mentioned in 1st part is used. Following is the Projection matrix obtained from DLT. Matrix is scaled to have a 1 in bottom right cell.

```
[[ 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]]
```

As we can see that results are not too far from the zhang's method implementation.

Python code for DLT implementation:

```
#Following code computes the projection matrix for given A matrix,
prints the calculated value for real world point (0, 0, 0) and
intrinsic parameters + extrinsic parameters matrix, values are
hardcoded for the given image and checker board's captured image.

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

margin = 36           #for the given image

#A = [[0,0,margin,1,0,0,0,0,0,0,-1*4731*margin,-4731], #Given Image
#[0,0,0,0,0,0,margin,1,0,0,-1*2453*margin,-2453],

#[0,margin,0,1,0,0,0,0,0,-1*4873*margin,0,-4873],
#[0,0,0,0,0,margin,0,1,0,-1*1407*margin,0,-1407],

#[0,2*margin,0,1,0,0,0,0,0,-1*4914*2*margin,0,-4914],
#[0,0,0,0,0,2*margin,0,1,0,-1*564*2*margin,0,-564],

#[margin,0,0,1,0,0,0,0,-1*4010*margin,0,0,-4010],
#[0,0,0,0,margin,0,0,1,-1*2073*margin,0,0,-2073],

#[0,0,0,1,0,0,0,0,0,0,0,-4826],
#[0,0,0,0,0,0,0,1,0,0,0,-2141],

#[0,0,2*margin,1,0,0,0,0,0,0,-1*4642*2*margin,-4642],
#[0,0,0,0,0,0,2*margin,1,0,0,-1*2800*2*margin,-2800]
#]

margin = 25 # for checkerboard
A = [[0,0,margin,1,0,0,0,0,0,0,-1*733*margin,-733], #Big ones
[0,0,0,0,0,0,margin,1,0,0,-1*989*margin,-989],

[0,margin,0,1,0,0,0,0,0,-1*792*margin,0,-792],
[0,0,0,0,0,margin,0,1,0,-1*821*margin,0,-821],

[0,2*margin,0,1,0,0,0,0,0,-1*775*2*margin,0,-775],
[0,0,0,0,0,2*margin,0,1,0,-1*698*2*margin,0,-698],

[margin,0,0,1,0,0,0,0,-1*912*margin,0,0,-912],
[0,0,0,0,margin,0,0,1,-1*950*margin,0,0,-950],

[0,0,0,1,0,0,0,0,0,0,0,-803],
[0,0,0,0,0,0,0,1,0,0,0,-939],

[0,0,2*margin,1,0,0,0,0,0,0,-1*651*2*margin,-651],
[0,0,0,0,0,0,2*margin,1,0,0,-1*1045*2*margin,-1045]
```

```
]



U,s,V = linalg.svd(A)
x = V[-1].reshape((3,4))
y = np.dot(x, [0,0,0,1])
print y[0]/y[2]
print y[1]/y[2]
M =[]
for i in x:
     M.append(i[0:3])

r, q = linalg.rq(M)
print r
print
print q
```