

CSE578: Assignment #1

Due on Thursday, February 4, 2016

Dr. Anoop Namboodiri

Abhinav Moudgil

20331039

Problem 1

Implement the DLT (Direct Linear Transformation) based calibration that we discussed in the class. You could use C/C++ or Matlab for this. However, you are expected to implement it yourselves and not use an existing implementation.

Procedure:

- A matrix is computed by appending two rows from each point in the matrix. Hence we get a matrix of dimension $2n \times 12$ from n points.
- Single value decomposition of A is computed.

$$A = UDV^T$$

- Column of V corresponding to smallest eigen value is picked.

$$x_i = P x_i$$

$$\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} U_i \\ V_i \\ W_i \\ T_i \end{bmatrix}$$

$$x = \begin{bmatrix} A^T \\ B^T \\ C^T \end{bmatrix} x_i$$

$$x_i = \frac{u_i}{w_i} = \frac{A^T x_i}{C^T x_i} \Rightarrow x_i C^T x_i - A^T x_i = 0$$

$$y_i = \frac{v_i}{w_i} = \frac{B^T x_i}{C^T x_i} \Rightarrow y_i C^T x_i - B^T x_i = 0$$

$$P = \begin{bmatrix} A \\ B \\ C \end{bmatrix}_{12 \times 1}$$

$$a_{x_i}^T P = 0$$

$$a_{y_i}^T P = 0$$

where,

$$a_{x_i}^T = (-X_i^T, 0^T, x_i X_i^T)_{1 \times 12}$$

$$a_{y_i}^T = (0^T, -X_i^T, y_i X_i^T)_{1 \times 12}$$

$$\begin{bmatrix} a^T x_1 \\ a^T y_1 \\ \vdots \\ a^T x_i \\ a^T y_i \\ \vdots \\ a^T x_n \\ a^T y_n \end{bmatrix}$$

$$P = M_{2n \times 12} \cdot P_{12 \times 1} = 0_{2n \times 1}$$

$$M \times P = w$$

Therefore, find P such that it maximizes

$$\Omega = w^T w$$

$$\Rightarrow \hat{P} = P w^T w = P P^T M^T M P$$

$$\|P\|_2 = \sum_{i,j} P_{ij}^2 = 1.$$

Equivalent to finding the null space of M . Solution \rightarrow Eigenvector to the smallest eigenvalue of M ,

$$M = U_{2n \times 12} \cdot S_{12 \times 12} \cdot V_{12 \times 12}^T$$

$$\begin{aligned} \Omega &= P^T M^T M P \\ &= P^T V S U^T U S V^T P \\ &= P^T V S^2 V^T P \\ &= P^T \left(\sum s^2 v_i v_i^T \right) P \end{aligned}$$

$$\hat{P} = \begin{pmatrix} \hat{A} \\ \hat{B} \\ \hat{C} \end{pmatrix} = V_{12}$$

No solution exists if all x_i 's on the same plane. Therefore, reduced rank of M ,

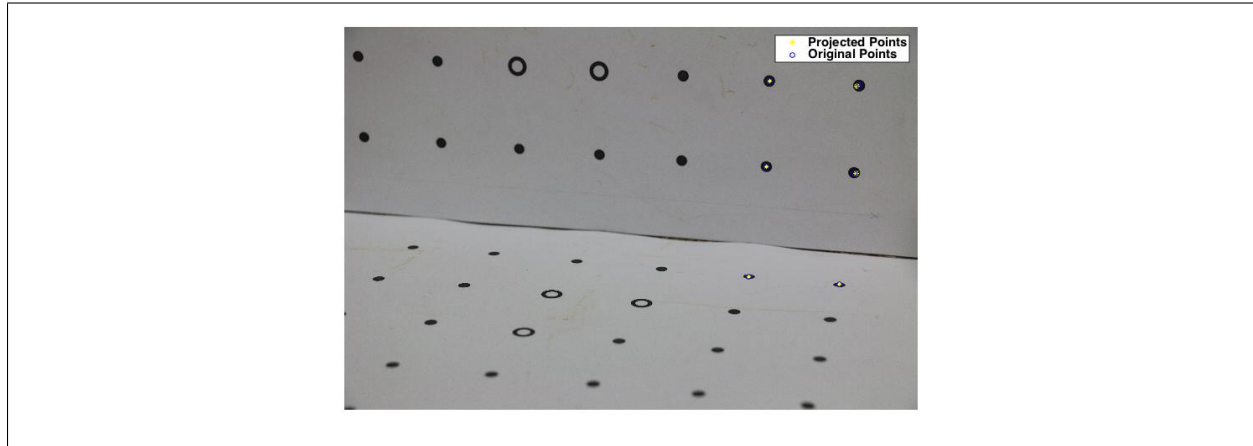
$$\hat{P} = \hat{K} \hat{R} (I_3 - \hat{X}_0) = (\hat{H}_\infty | \hat{h})$$

$$\hat{X}_0 = -\hat{H}_\infty^{-1} \hat{h}$$

$$\hat{H}_\infty = \hat{K} \hat{R}$$

where \hat{R} is rotation matrix and \hat{K} is upper triangular matrix.
Fundamental Camera Matrix P by DLT:

$$\begin{bmatrix} 0.0043 & -0.0008 & 0.0003 & -0.9062 \\ 0.0002 & 0.0043 & -0.0012 & -0.4228 \\ -0.0000 & -0.0000 & -0.0000 & -0.0002 \end{bmatrix}$$



```

1 clear all;
2 im = imread('IMG_5455.JPG');
3 szx = size(im, 1);
4 szy = size(im, 2);
5
6 x = [4.917905405405405e+03;4.868608108108108e+03;4.726878378378378e+03;4.061364864864866e
      +03;4.030554054054055e+03;3.864175675675676e+03];
7 y = [5.581756756756758e+02;1.396229729729730e+03;2.456121621621622e+03;5.150405405405404e
      +02;1.334608108108108e+03;2.382175675675676e+03];
8
9 X = [0,0,0,36,36,36];
10 Y = [72, 36, 0, 72, 36, 0];
11 Z = [0,0,36,0,0,36];
12
13 XI = [x';y';ones(6,1)'];
14 XW = [X;Y;Z;ones(6,1)'];
15
16 P = fdlr(XW,XI)
17 projected = P*XW;
18
19 projected(1,:) = projected(1,:)/projected(3,:);
20 projected(2,:) = projected(2,:)/projected(3,:);
21 imshow(im);
22 hold on;
23 plot(projected(1,:), projected(2,:), 'y*');
24 plot(x,y, 'r*');
25
26 figure, imshow(im);
27 hold on;
28 plot(projected(1,:), projected(2,:), 'r*');
29
30 figure, imshow(im);
31 hold on;
32 plot(x, y, 'y*');

```

```

1 function P = fdlr(xw, xi)
2
3     x = xi(1,:);
4     y = xi(2,:);
5     z = xi(3,:);
6     n = size(xw,2);
7     A = zeros(2*n,12);
8     for i = 1 : n
9         A(2*i-1:2*i, :) = [-z(i)*xw(:,i)' zeros(1,4) x(i)*xw(:,i)';
10                          zeros(1,4) -z(i)*xw(:,i)' y(i)*xw(:,i)'];

```

```
11     end
12
13     for i = 1 : 3
14         A(i, :) = A(i, :) / norm(A(i, :));
15
16         [u, s, v] = svd(A);
17         P = reshape(v(:, 12), [4, 3])';
18     end
```

Problem 2

Implement the RANSAC based 8. 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 is a resampling technique that generates candidate solutions by using the minimum number observations (data points) required to estimate the underlying model parameters. As pointed out by Fischler and Bolles, unlike conventional sampling techniques that use as much of the data as possible to obtain an initial solution and then proceed to prune outliers, RANSAC uses the smallest set possible and proceeds to enlarge this set with consistent data points.

The number of iterations, N , is chosen high enough to ensure that the probability p (usually set to 0.99) that at least one of the sets of random samples does not include an outlier. Let u represent the probability that any selected data point is an inlier and $v = 1 - u$ the probability of observing an outlier. N iterations of the minimum number of points denoted m are required, where

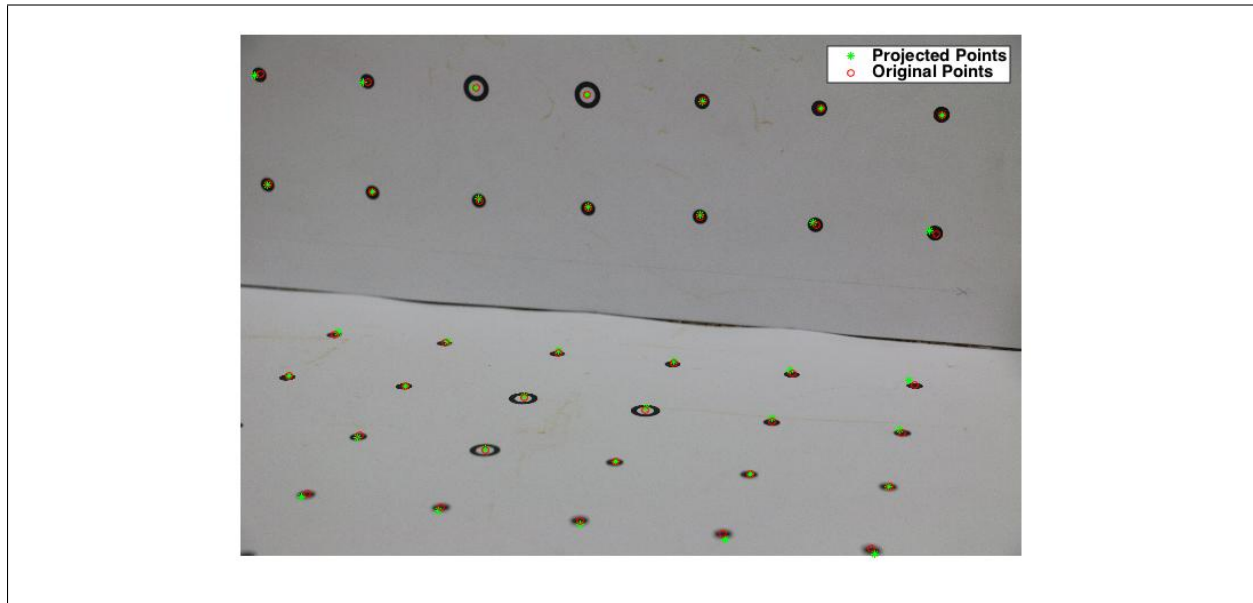
$$1 - p = (1 - u^m)^N$$

and thus with some manipulation,

$$N = \frac{\log(1 - p)}{\log(1 - (1 - u)^m)}$$

Considering 0.75 to be the probability of a point being an outlier, we get $N = 20$.
Fundamental Camera Matrix P by RANSAC:

$$\begin{bmatrix} 0.0040 & 0.0005 & 0.0018 & -0.9113 \\ 0.0002 & 0.0043 & -0.0008 & -0.4116 \\ -0.0000 & 0.0000 & -0.0000 & -0.0002 \end{bmatrix}$$



```

1  close all;
2
3  clc;
4  im = imread('IMG_5455.JPG');
5
6  % imshow(im);
7  % [x, y] = ginput();
8
9  x = [4.916500e+03 4.068500e+03 3.244500e+03 2.428500e+03 1.652500e+03 8.925000e+02 1.405000e
+02 4.876500e+03...
10  4.036500e+03 3.220500e+03 2.436500e+03 1.676500e+03 9.245000e+02 1.965000e+02 4.724500e
+03 3.868500e+03...
11  3.036500e+03 2.228500e+03 1.436500e+03 6.605000e+02 4.636500e+03 3.724500e+03 2.836500e
+03 1.988500e+03...
12  1.156500e+03 3.405000e+02 4.548500e+03 3.572500e+03 2.628500e+03 1.716500e+03 8.365000e
+02 4.420500e+03...
13  3.380500e+03 2.380500e+03 1.404500e+03 4.685000e+02];
14  y = [5.645000e+02 5.165000e+02 4.605000e+02 4.205000e+02 3.725000e+02 3.325000e+02 2.765000e
+02 1.396500e+03...
15  1.332500e+03 1.276500e+03 1.212500e+03 1.164500e+03 1.100500e+03 1.044500e+03 2.452500e
+03 2.372500e+03...
16  2.300500e+03 2.228500e+03 2.156500e+03 2.100500e+03 2.788500e+03 2.708500e+03 2.628500e
+03 2.540500e+03...
17  2.460500e+03 2.388500e+03 3.164500e+03 3.076500e+03 2.988500e+03 2.908500e+03 2.804500e
+03 3.596500e+03...
18  3.492500e+03 3.404500e+03 3.308500e+03 3.212500e+03];
19
20  X = [0:36:216 0:36:216 0:36:180 0:36:180 0:36:144 0:36:144];
21  Y = [72*ones(7,1)' 36*ones(7,1)' zeros(22,1)'];
22  Z = [zeros(14,1)' 36*ones(6,1)' 72*ones(6,1)' 108*ones(5,1)' 144*ones(5,1)'];
23
24  N = 5;
25
26  theta = 1;
27
28  max = 0;
29  XI = [x;y;ones(36,1)'];
30  XW = [X;Y;Z;ones(36,1)'];
31
32  for i = 1 : 150

```

```

33     r = [randi([1 14], 1, 3) randi([15 36], 1, 3)];
34     P = fdlr(XW(:, r), XI(:, r));
35     projected = P * XW;
36     projected_x = projected(1, :)/projected(3, :);
37     projected_y = projected(2, :)/projected(3, :);
38     error_x = (abs(projected_x - x));
39     error_y = (abs(projected_y - y));
40     error = error_x + error_y;
41     t = error < theta;
42     number_of_inliers = sum(t);
43     if (number_of_inliers > max)
44         max = number_of_inliers;
45         solution = P;
46     end
47 end
48
49 projected = solution * XW;
50 projected(1, :) = projected(1, :)/projected(3, :);
51 projected(2, :) = projected(2, :)/projected(3, :);
52 imshow(im);
53 hold on;
54 plot(projected(1,:), projected(2,:), 'y*');
55 plot(x,y, 'r*');

```

Problem 3

Use the image that is provided along with this assignment along with the 9. real-world measurements that are given to you to compute the cameras internal and external parameters using each of the above algorithms. Note that you need to manually estimate the image co-ordinates of the given world points. You may implement and use the 10. Harris Corner detector to help select these points. Describe your observations regarding the results.

$$A = RQ$$

R: Rotation matrix

Q: Upper triangular matrix

$$\hat{H}_{\infty}^{-1} = \hat{R}^T \cdot \hat{K}^{-1}$$

Normalize K

$$\hat{K} = \frac{1}{\hat{K}_{33}} \hat{K}$$

$$k = \begin{bmatrix} -1.4574 & -0.0112 & -0.0726 \\ 0 & -1.4592 & 0.1060 \\ 0 & 0 & 0.0001 \end{bmatrix} * 10^4$$

$$R = \begin{bmatrix} -0.9587 & -0.0057 & -0.2845 \\ -0.0965 & -0.9341 & 0.3437 \\ -0.2677 & 0.3569 & 0.8950 \end{bmatrix}$$

$$t = \begin{bmatrix} -55.3688 \\ 199.8165 \\ 615.0012 \end{bmatrix}$$

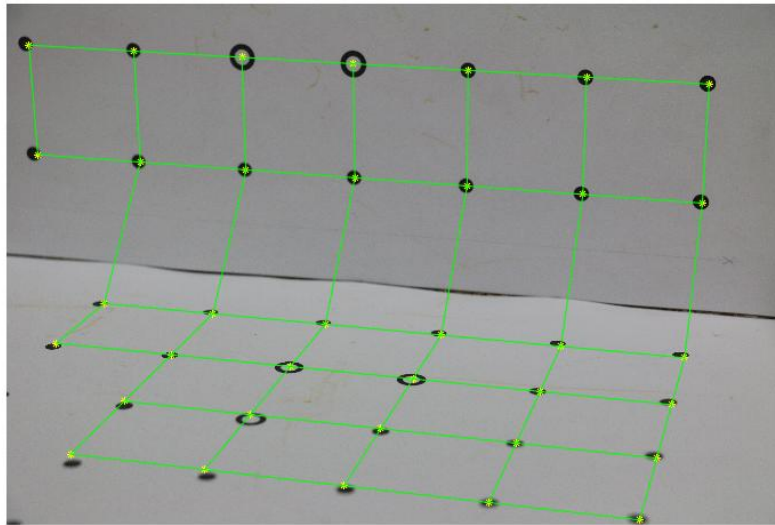
```

1 [Q, W] = qr(inv(P(1:3, 1:3)));
2
3 % P = [KR -KRC]
4
5 % Rotation Matrix
6 R = inv(Q);
7
8 % Intrinsic Camera Matrix
9 K = inv(W);
10 K = K/K(3,3);
11
12 % Camera Center
13 C = - (inv(R) * inv(K) * P(:, 4));

```

Problem 4

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 $x_i = P.X_i$, and not use the image points. Overlay (draw) the wireframe over the actual image of the object using straight lines between the computed points x_i . What do you observe about the overlay?



```

1 close all;
2 clear;
3 clc;
4 im = imread('IMG_5455.JPG');
5 % [x, y] = ginput();
6
7 x = [4.916500e+03 4.068500e+03 3.244500e+03 2.428500e+03 1.652500e+03 8.925000e+02 1.405000e+02 4.876500e+03 4.036500e+03 3.220500e+03 2.436500e+03 1.676500e+03 9.245000e+02 1.965000e+02 4.724500e+03 3.868500e+03 3.036500e+03 2.228500e+03 1.436500e+03 6.605000e+02 4.636500e+03 3.724500e+03 2.836500e+03 1.988500e+03 1.156500e+03 3.405000e+02 4.548500e+03 3.572500e+03 2.628500e+03 1.716500e+03 8.365000e+02 4.420500e+03 3.380500e+03 2.380500e+03 1.404500e+03 4.685000e+02];
8 y = [5.645000e+02 5.165000e+02 4.605000e+02 4.205000e+02 3.725000e+02 3.325000e+02 2.765000e+02 1.396500e+03 1.332500e+03 1.276500e+03 1.212500e+03 1.164500e+03 1.100500e+03

```



```

1.044500e+03 2.452500e+03 2.372500e+03 2.300500e+03 2.228500e+03 2.156500e+03 2.100500e
+03 2.788500e+03 2.708500e+03 2.628500e+03 2.540500e+03 2.460500e+03 2.388500e+03
3.164500e+03 3.076500e+03 2.988500e+03 2.908500e+03 2.804500e+03 3.596500e+03 3.492500e
+03 3.404500e+03 3.308500e+03 3.212500e+03];
9
10 X = [0:36:216 0:36:216 0:36:180 0:36:180 0:36:144 0:36:144];
11 Y = [72*ones(7,1)' 36*ones(7,1)' zeros(22,1)'];
12 Z = [zeros(14, 1)' 36*ones(6,1)' 72*ones(6,1)' 108*ones(5,1)' 144*ones(5,1)'];
13
14 N = 250;
15
16 theta = 1;
17
18 max = 0;
19 XI = [x;y;ones(36,1)'];
20 XW = [X;Y;Z;ones(36,1)'];
21
22 for i = 1 : N
23     r = [randi([1 14], 1, 3) randi([15 36], 1, 3)];
24     P = fdlr(XW(:, r), XI(:, r));
25     projected = P * XW;
26     projected_x = projected(1, :)/projected(3, :);
27     projected_y = projected(2, :)/projected(3, :);
28     error_x = (abs(projected_x - x));
29     error_y = (abs(projected_y - y));
30     error = error_x + error_y;
31     t = error < theta;
32     number_of_inliers = sum(t);
33     if (number_of_inliers > max)
34         max = number_of_inliers;
35         solution = P;
36     end
37 end
38
39 projected = solution * XW;
40 projected(1, :) = projected(1, :)/projected(3, :);
41 projected(2, :) = projected(2, :)/projected(3, :);
42
43 figure, imshow(im);
44 hold on;
45 plot(projected(1,:), projected(2,:), 'y*');
46 plot(x, y, 'r*');
47 hold on;
48
49 for i = 2:36
50     if (i ~= 8 && i ~= 15 && i ~= 21 && i ~= 27 && i ~= 32)
51         plot([projected(1, i), projected(1, i-1)], [projected(2, i), projected(2, i-1)], 'g');
52     end
53 end
54
55 for i = 1:7
56     plot([projected(1, i), projected(1, 7+i)], [projected(2, i), projected(2, 7+i)], 'g'
57 );
58     if (i <= 6)
59         plot([projected(1, 7+i), projected(1, 14+i)], [projected(2, 7+i), projected(2, 14+i)
60 ], 'g');
61         plot([projected(1, 14+i), projected(1, 20+i)], [projected(2, 14+i), projected(2, 20+
62 i)], 'g');
63     end
64     if (i <= 5)
65         plot([projected(1, 20+i), projected(1, 26+i)], [projected(2, 20+i), projected(2, 26+
66 i)], 'g');
67     end

```

```

63     plot([projected(1, 26+i), projected(1, 31+i)], [projected(2, 26+i), projected(2, 31+
        i)], 'g');
64     end
65 end

```

Problem 5

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

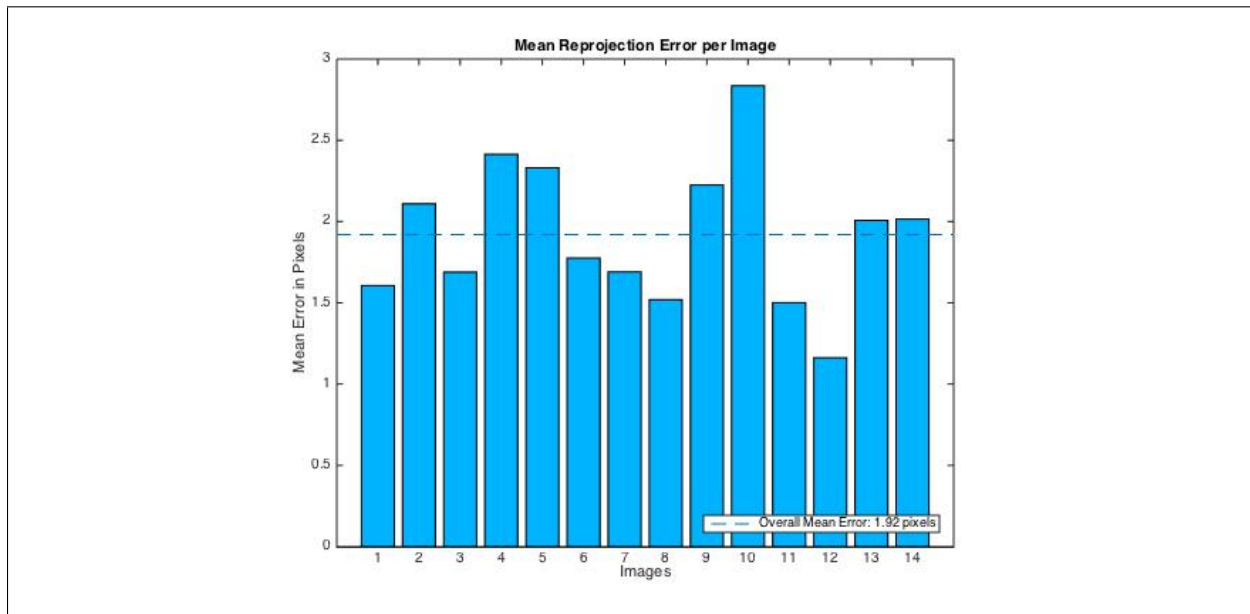
We use a checkboard pattern for camera calibration in this method. The pattern is placed on model plane (Z component in world coordinate is zero) as illustrated below. Several pictures are taken by changing the location and rotation angle. Camera parameters are estimated by Zhang's method as described his research paper.

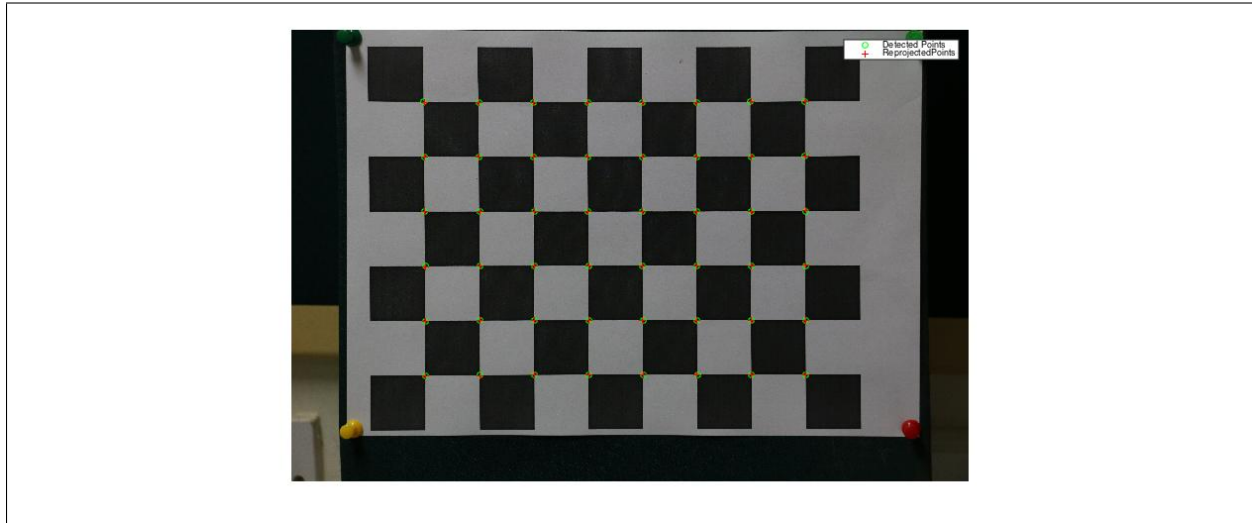
Estimated camera matrix K' :

$$1.0e + 04 * \begin{bmatrix} 1.3478 & 0 & 0 \\ 0 & 1.3531 & 0 \\ 0.2750 & 0.1836 & 0.0001 \end{bmatrix}$$

Radial distortion parameters:

$$\begin{bmatrix} 0.2849 \\ -0.7455 \end{bmatrix}$$





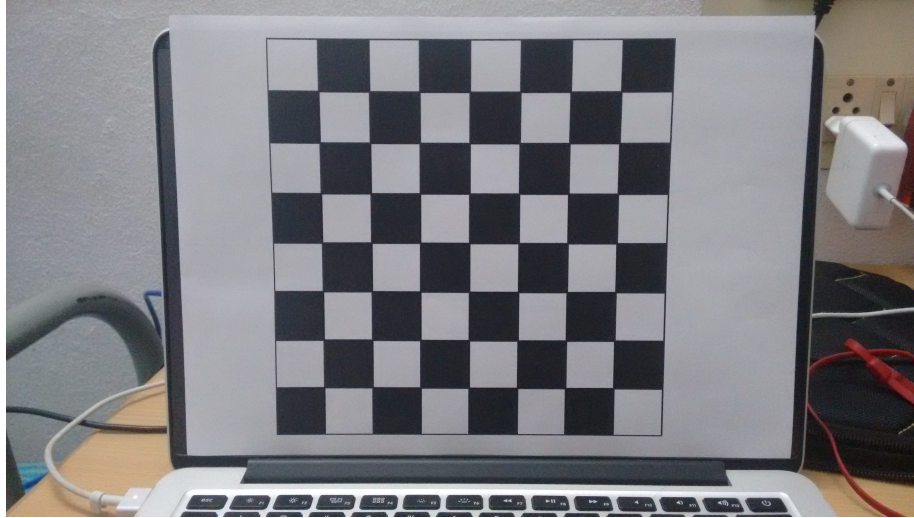
```

1 clear;
2 close all;
3 clc;
4
5 location = '/Users/abhinavmoudgil95/Documents/MATLAB/CV/assignment1/images/';
6 images = imageSet(location);
7 imageFileNames = images.ImageLocation;
8 [imagePoints, boardSize] = detectCheckerboardPoints(imageFileNames);
9 squareSizeInMM = 29;
10 worldPoints = generateCheckerboardPoints(boardSize, squareSizeInMM);
11 params = estimateCameraParameters(imagePoints, worldPoints);
12 showReprojectionErrors(params);
13
14 figure;
15 imshow(imageFileNames{1});
16 hold on;
17 plot(imagePoints(:,1,1), imagePoints(:,2,1), 'go');
18 plot(params.ReprojectedPoints(:,1,1), params.ReprojectedPoints(:,2,1), 'r+');
19 legend('Detected Points', 'ReprojectedPoints');
20 hold off;

```

Problem 6

Repeat the 3 calibration methods using your own camera and your own calibration object for which you measure and determine the world co-ordinates. Use a printed checkerboard pattern for Zhangs method. Describe and comment on the results of each method.

Checkerboard**Measurements**