COMPUTER VISION

ASSIGNMENT 1

Answer 1) DLT method is used for camera calibration using image of a 3D object in 3D world coordinates. In camera calibration, we try to estimate camera intrinsic parameters (focal lengths, skew, principal point) and extrinsic (rotation matrix and translation vector).

DLT Algorithm:

1) Take a *n* number of 2D images points, say $\mathbf{x_i}$ and corresponding 3D world points, say $\mathbf{X_i}$, where

$$\boldsymbol{x}_\mathbf{i} = \begin{bmatrix} \mathrm{x_i} \\ \mathrm{y_i} \end{bmatrix} \quad \text{and} \quad \mathbf{X_i} = \begin{bmatrix} \mathrm{X_i} \\ \mathrm{Y_i} \\ \mathrm{Z_i} \end{bmatrix}$$

2) Select any 6 non-coplanar 2D image points from the above set. Take the corresponding 3D world points as well.

3) Create a 12x12 matrix of the following form,

$$\begin{bmatrix} \mathrm{X_i} & \mathrm{Y_i} & \mathrm{Z_i} & 1 & 0 & 0 & 0 & 0 & -\mathrm{x_i} * \mathrm{X_i} & -\mathrm{x_i} * \mathrm{Y_i} & -\mathrm{x_i} * \mathrm{Z_i} & -\mathrm{x_i} \\ 0 & 0 & 0 & 0 & \mathrm{X_i} & \mathrm{Y_i} & \mathrm{Z_i} & 1 & -\mathrm{y_i} * \mathrm{X_i} & -\mathrm{y_i} * \mathrm{Y_i} & -\mathrm{y_i} * \mathrm{Z_i} & -\mathrm{y_i} \end{bmatrix}$$

Each i^{th} point correspondence contributes to 2 such rows. So, for 6 points, the matrix is 12x12.

4) Solve the above matrix using SVD and take the vector corresponding to the smallest eigen value.

5) Reshape the vector obtained in step 4, to get the 3x4 projection matrix, *P*.

6) Using QR decomposition of $P(:,1:3)^{-1}$ matrix, obtain R^{-1} and K^{-1}. Using the $K^{-1} * P(:,4)$, obtain value of *t.*

Challenges: Major challenge was to obtain the center points of the given circles and disks. Instead of doing it completely manually which will involve higher error in positions, I first applied morphological operations to convert disk into circles and to reduce the circles to points. Then used Harris corner detector to give coordinates of those points. Still for some I got two or more corners representing the same point. Selecting one of those coordinates was done manually.

Results:

For image '`IMG_5455.JPG`',

Calibration Matrix :

```
K_1 =

  1.0e+04 *

  -1.1141    0.0104    0.2602
        0   -1.1031    0.4013
        0         0    0.0001
```

After correction for negative focal length

```
K_1 =

  1.0e+04 *

   1.1141   -0.0104    0.2602
        0    1.1031    0.4013
        0         0    0.0001
```

Rotation Matrix :

```
R_1 =

   -0.9707   -0.0000   -0.2405
   -0.1194   -0.8680    0.4820
   -0.2087    0.4965    0.8426
```

After correction for negative focal length

```
R_1 =

    0.9707    0.0000    0.2405
    0.1194    0.8680   -0.4820
   -0.2087    0.4965    0.8426
```

Translation Vector :

```
t_1 =

   96.1769
  -82.8977
 -495.3977
```

After correction for negative focal length

```
t_1 =

  -96.1769
   82.8977
 -495.3977
```

Code:

```
I = imread('IMG_5455.JPG');
I_g = rgb2gray(I);

% Image preprocessing
I_b = imbinarize(I_g,0.3);
I_m = imdilate(I_b,strel('rectangle',[17,90]));

% Detecting points using Harris Detector
corners = detectHarrisFeatures(I_m);
n = 45;
a = corners.selectStrongest(n);
loc = a.Location();
```

```
% LUT for 2D and 3D points
lut2D = [133,271,1; 895,339,1;1651,375,1; 2433,429,1; 3236,450,1;...
         4059,495,1;4917,536,1;194,1049,1;929,1107,1;1672,1159,1;...
         2437,1209,1;3222,1263,1;4032,1351,1;4870,1414,1;660,2094,1;...
         1435,2155,1;2223,2236,1;3028,2308,1;3860,2379,1;4721,2458,1;...
         326,2405,1;1157,2461,1;1981,2546,1;2838,2627,1;3732,2717,1;...
         4635,2790,1; 821,2821,1;1705,2910,1;2620,2996,1;3570,3081,1;...
         4539,3166,1; 467,3220,1;1411,3311,1;2382,3402,1;3384,3498,1];
lut3D = [216,72,0,1; 180,72,0,1; 144,72,0,1; 108,72,0,1; 72,72,0,1;...
         36,72,0,1; 0,72,0,1; 216,36,0,1; 180,36,0,1; 144,36,0,1;...
         108,36,0,1; 72,36,0,1; 36,36,0,1; 0,36,0,1; 180,0,36,1;...
         144,0,36,1; 108,0,36,1; 72,0,36,1; 36,0,36,1; 0,0,36,1;...
         180,0,72,1; 144,0,72,1; 108,0,72,1; 72,0,72,1; 36,0,72,1;...
         0,0,72,1; 144,0,108,1; 108,0,108,1; 72,0,108,1; 36,0,108,1;...
         0,0,108,1; 144,0,144,1; 108,0,144,1; 72,0,144,1; 36,0,144,1];
```

```matlab
% DLT
x = lut2D([1,15,7,12,24,17],:);
X = lut3D([1,15,7,12,24,17],:);

P_1 = DLT(x,X);
[K_1, R_1, t_1] = decompose_P(P_1);

% Reprojection Error
genPts2D = (P_1 * lut3D)';
genPts2D = genPts2D./genPts2D(:,3);
err = (lut2D - genPts2D).^2;
err = sum(sqrt(sum(err(:,1:2), 2)));

function P = DLT(x, X)

      len = length(x);

      % Projection Matrix
      A = [ X(:,1), X(:,2), X(:,3), X(:,4), zeros(len,4),...
            -x(:,1).*X(:,1), -x(:,1).*X(:,2), -x(:,1).*X(:,3), -x(:,1);...
      zeros(len,4), X(:,1), X(:,2), X(:,3), X(:,4), -x(:,2).*X(:,1),...
      -x(:,2).*X(:,2), -x(:,2).*X(:,3), -x(:,2)];

      [~,~,V] = svd(A);
      P = reshape(V(:,end),[4,3])';

end

function [K, R, t] = decompose_P(P)

      % Decomposes Projection matrix to K, R, t
      KR = [P(:,1), P(:,2), P(:,3)];
      inv_KR = inv(KR);

      [R, K] = qr(inv_KR);

      R = R';
      t = K*P(:,4);

      % Normalize K matrix
      K = inv(K);
      K = K./K(3,3);
end
```

Answer 2) DLT using Ransac randomly selects 6 corresponding points i.e., 2D and 3D for *itr* number of iterations. For each iteration, it calculates projection matrix and calculates reprojection error in each iteration. After completing all the iteration, it returns the projection matrix which has least error.

Algorithm:

1) Set *itr*, *errThr*.

2) For each *itr*, choose non-coplanar 6 image points, **x** and the corresponding 3D points, **X**.

3) Calculate projection matrix, *P*, using the steps 3-4 mentioned in the algorithm of answer 1.

4) Generate reprojected image points using the *P* and all the 3D world points.

5) Store the number of inliers which have reprojection error less than the *errThr* along with *P* matrix.

6) After all iterations, find the *P* matrix with maximum number of inliers.

7) Decompose *P* matrix into *K*, *R* and *t* using the step-6 of answer 1.

Results:

For image 'IMG_5455.JPG',
Calibration Matrix:

```
K_2 =

  1.0e+04 *

  -1.2933    0.0241    0.2760
        0   -1.3027    0.2486
        0         0    0.0001
```

After correction for negative focal length

```
K_2 =

  1.0e+04 *

   1.2933   -0.0241    0.2760
        0    1.3027    0.2486
        0         0    0.0001
```

Rotation Matrix:

```
R_2 =

   0.9999    0.0033    0.0127
  -0.0117    0.6571    0.7537
  -0.0059   -0.7538    0.6571
```

After correction for negative focal length

```
R_2 =

  -0.9999   -0.0033   -0.0127
   0.0117   -0.6571   -0.7537
  -0.0059   -0.7538    0.6571
```

Translation Vector:

```
t_2 =

  -50.6817
  -31.2443
  162.3140
```

After correction for negative focal length

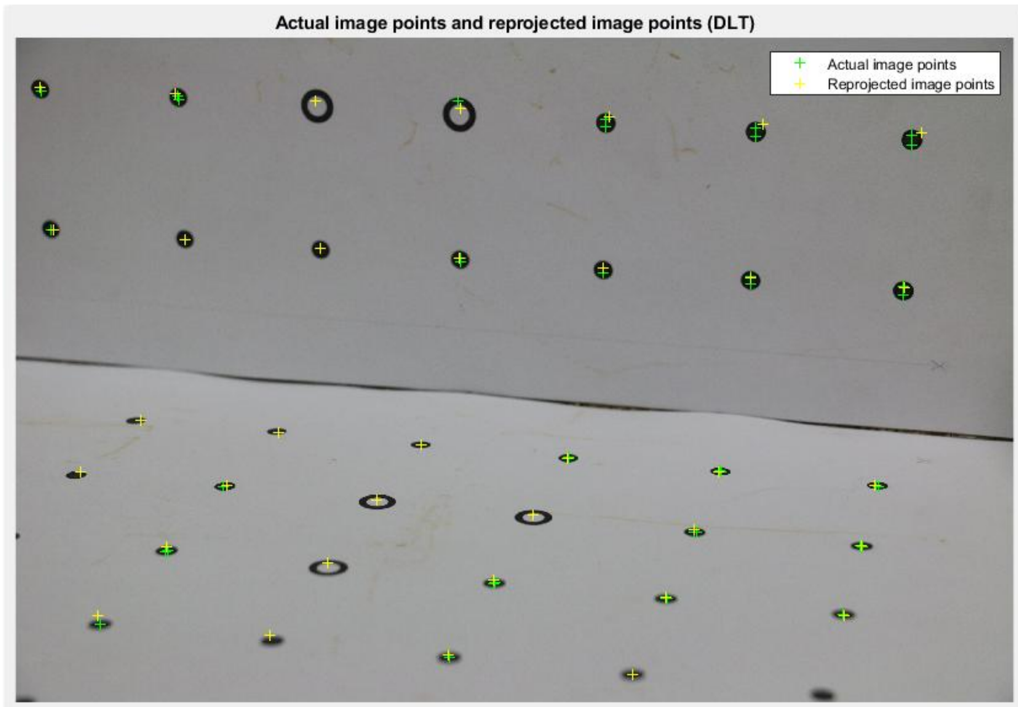
```
t_2 =

   50.6817
   31.2443
 -162.3140
```

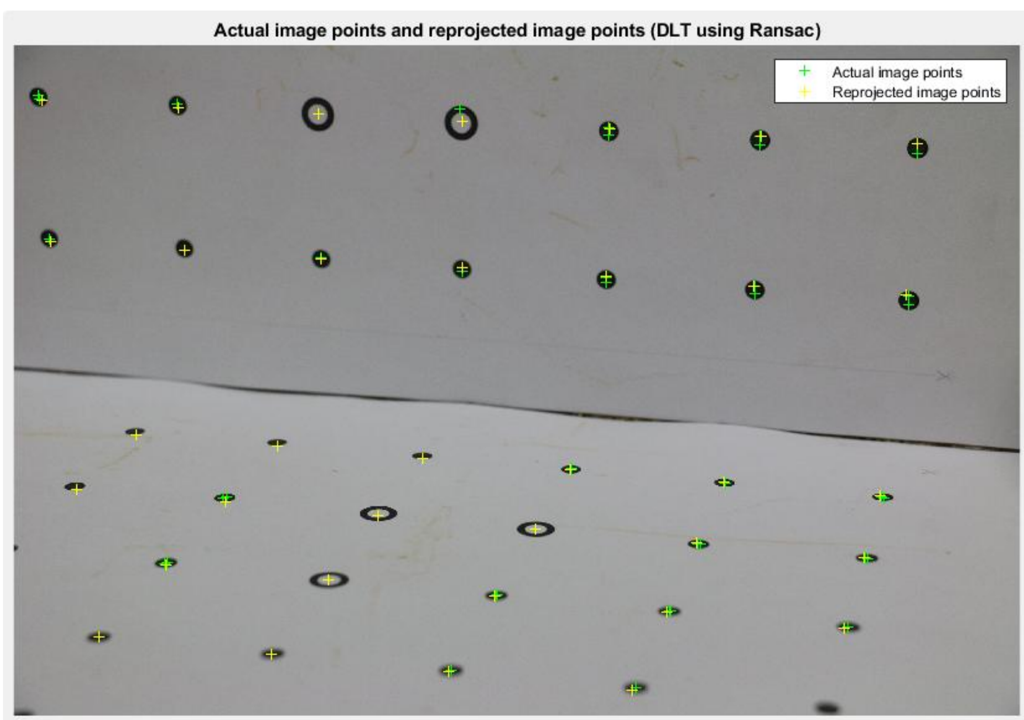
Code:

```
pts2D = lut2D;
pts3D = lut3D;
itrs = 250;
n = 6;
errThr = 5;
p = ones(itrs, 3, 4);
inliers = ones(itrs, 1);
for itr = 1:itrs
      % Select points randomly
      idx = randperm(numel(1:size(pts2D, 1)));
      pts = idx(1:n);
      p(itr, :, :) = DLT(pts2D(pts,:), pts3D(pts,:));

      % Calculate reprojection error
      genPts2D = (reshape(p(itr,:,:), [3, 4]) * pts3D')';
      err = pts2D - genPts2D;
      err = err.^2;
      err = sqrt(sum(err(:,1:2), 2));

      inliers(itr) = size(find(err < errThr),1);
end

% Find best intrinsic and extrinsic parameters
[~, max_index] = max(inliers);

P_2 = reshape(p(max_index, :, :), [3, 4]);
[K_2, R_2, t_2] = decompose_P(P_2);

% Reprojection Error
genRansacPts2D = (P_2 * lut3D)';
genRansacPts2D = genRansacPts2D./ genRansacPts2D(:,3);
errRansac = (lut2D - genRansacPts2D).^2;
errRansac = sum(sqrt(sum(errRansac (:,1:2), 2)));
```

Answer 3) The projection matrix obtained using RANSAC gives less reprojection error. But RANSAC's basically depends on the set of points randomly picked by the algorithm in each iteration. If the points chosen are outliers or coplanar, then the probability of getting undesirable results increases.

Results : Reprojection error in DLT = 410(approx) and Reprojection error in Ransac = 345(approx). Error is square root of SSD.





Code:

```
I = imread('IMG_5455.JPG');
I_g = rgb2gray(I);

% Image preprocessing
I_b = imbinarize(I_g, 0.3);
I_m = imdilate(I_b,strel('rectangle', [17,90]));

% Detecting points using Harris Detector
corners = detectHarrisFeatures(I_m);
n = 45;
a = corners.selectStrongest(n);
loc = a.Location();

% Reprojected points using P matrix obtained using DLT
genPts2D = (P * lut3D')';
genPts2D = genPts2D./genPts2D(:,3);

imshow(I); hold on; plot(a);
hold on;
plot(genPts2D(:,1), genPts2D(:,2), '+y');

% Reprojected points using P matrix obtained using DLT using Ransac
genRansacPts2D = (P_2 * lut3D')';
genRansacPts2D = genRansacPts2D./genRansacPts2D(:,3);

figure;
imshow(I); hold on; plot(a);
hold on;
plot(genRansacPts2D(:,1), genRansacPts2D(:,2), '+y');
```

Answer 4) Radial distortion formula,

$$x_{\text{distorted}} = x(1 + k_1{}^*r^2 + k_2{}^*r^4 + k_3{}^*r^6)$$

$$y_{\text{distorted}} = y(1 + k_1{}^*r^2 + k_2{}^*r^4 + k_3{}^*r^6)$$

where,

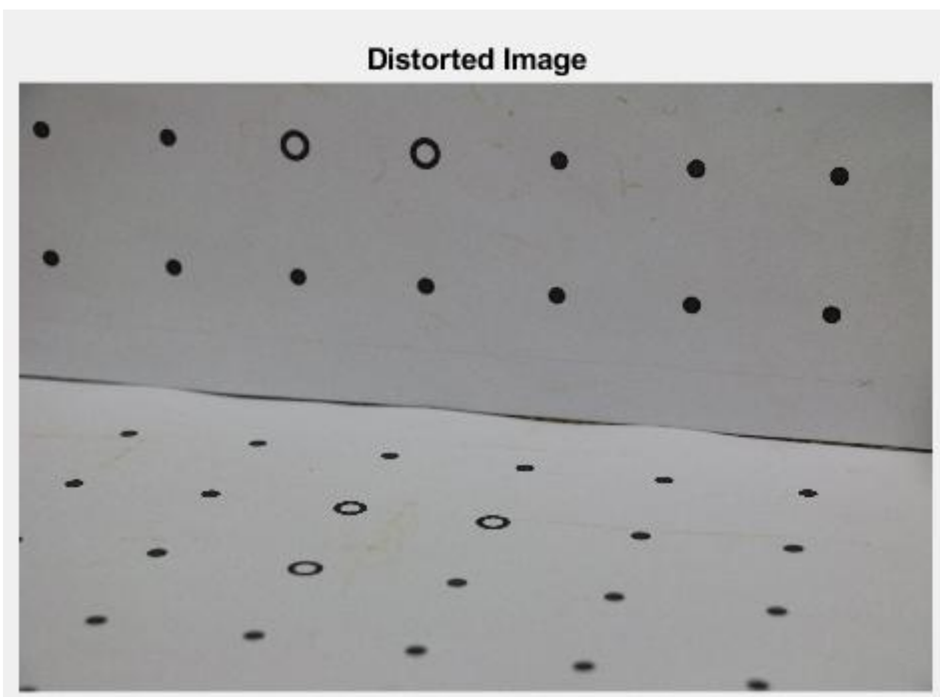
x,y = Undistorted pixel locations

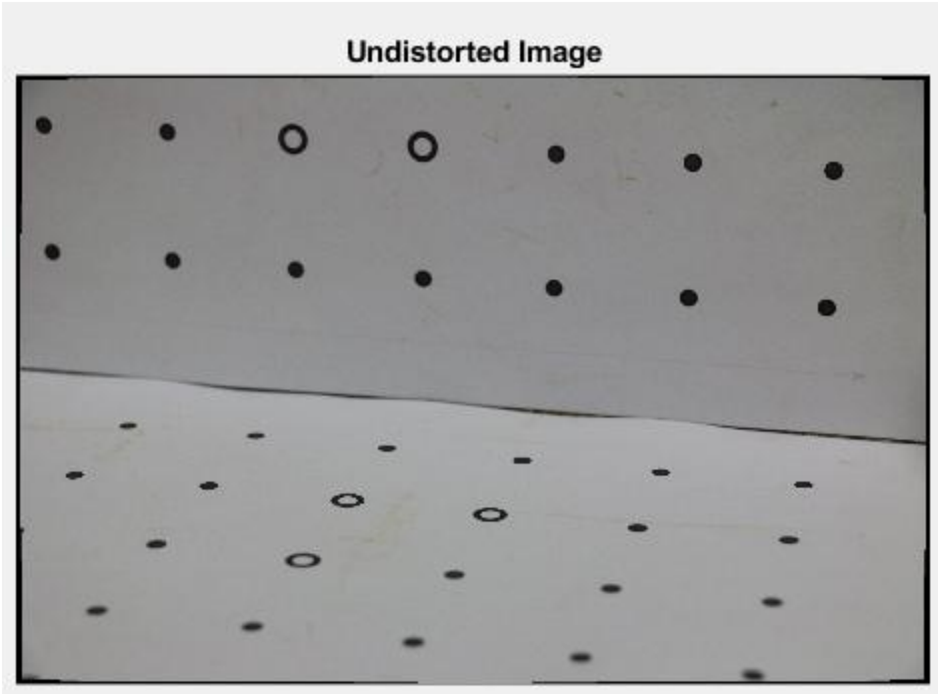
r = sqrt(x^2 + y^2)

k_1, k_2, k_3 = Radial distortion coefficients of the lens

Challenge: In order to correctly obtain the coefficients, the camera parameters object obtained using inbuilt Zhang's method was used. Assuming that all images provided are taken using same camera, the same coefficients were used to undistort the images.

Results: Radial distortion parameter obtained are k_1 = 0.2902 and k_2 = -0.9856





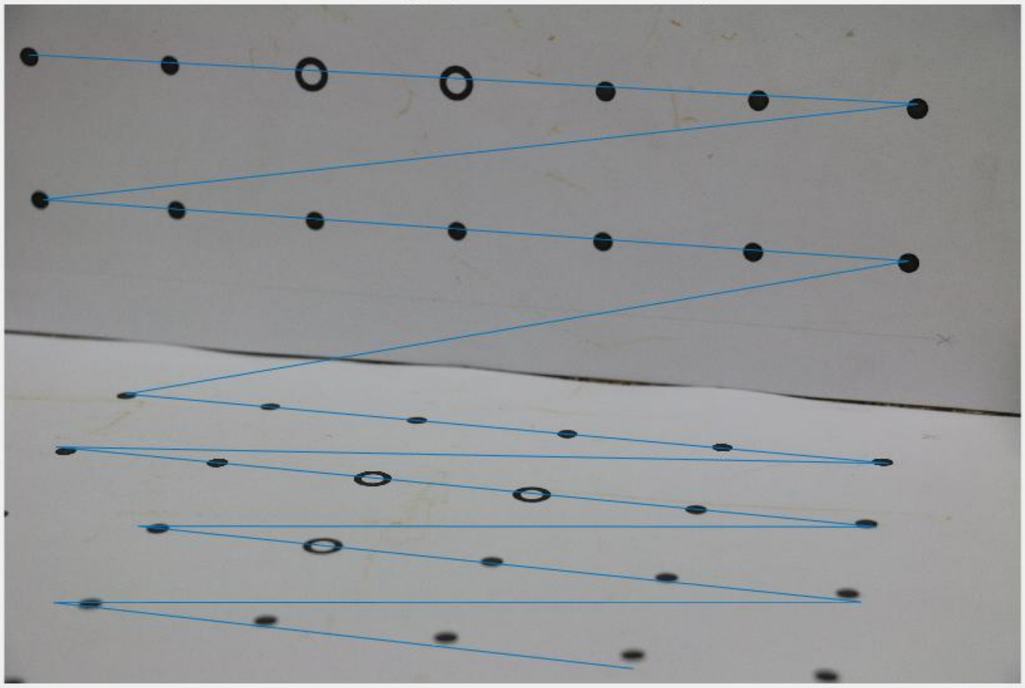
Code:

```
I = imread('IMG_5455.JPG');
I = undistortImage(I, cameraParams);

I_g = rgb2gray(I);

% Image preprocessing
I_b = imbinarize(I_g,0.3);
I_m = imdilate(I_b,strel('rectangle',[17,90]));

% Detecting points using Harris Detector
corners = detectHarrisFeatures(I_m);
n = 45;
a = corners.selectStrongest(n);
loc = a.Location();

% DLT
x = lut2D([1,15,7,12,24,17],:);
X = lut3D([1,15,7,12,24,17],:);

P_1 = DLT(x,X);
[K_1, R_1, t_1] = decompose_P(P_1);

% Reprojection Error
genPts2D = (P_1 * lut3D)';
genPts2D = genPts2D./genPts2D(:,3);
err = (lut2D - genPts2D).^2;
err = sum(sqrt(sum(err(:,1:2), 2)));
```

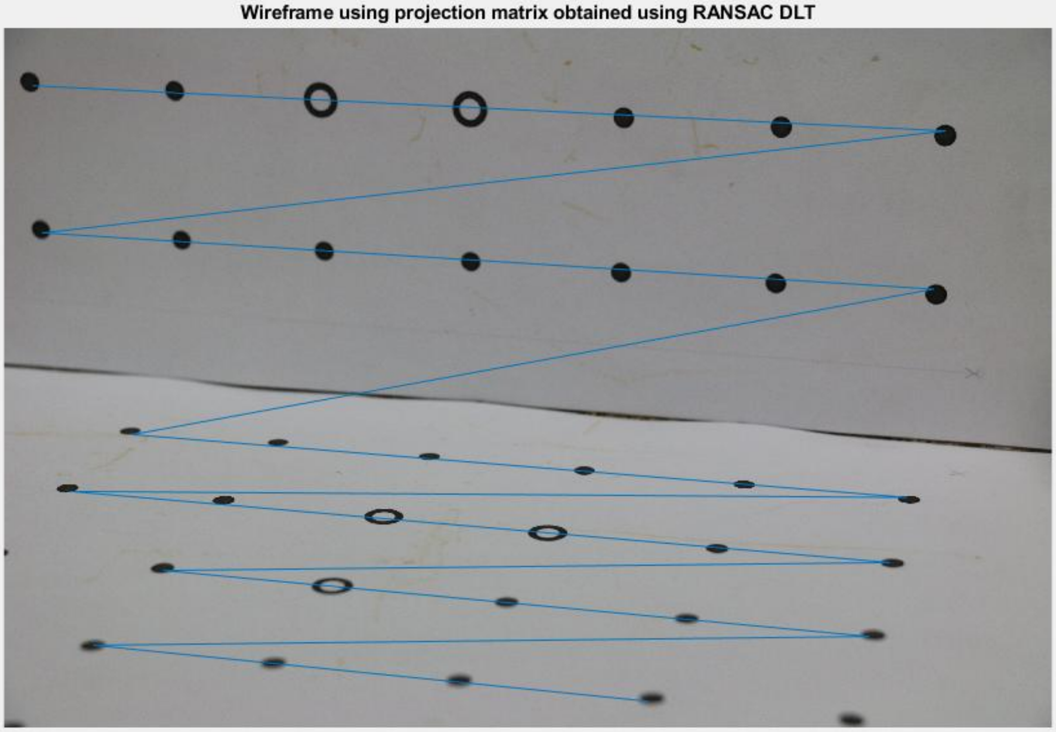
Answer 5) For generating the wireframes, all the 3D points are pre-multiplied with the projection matrix.

Results:

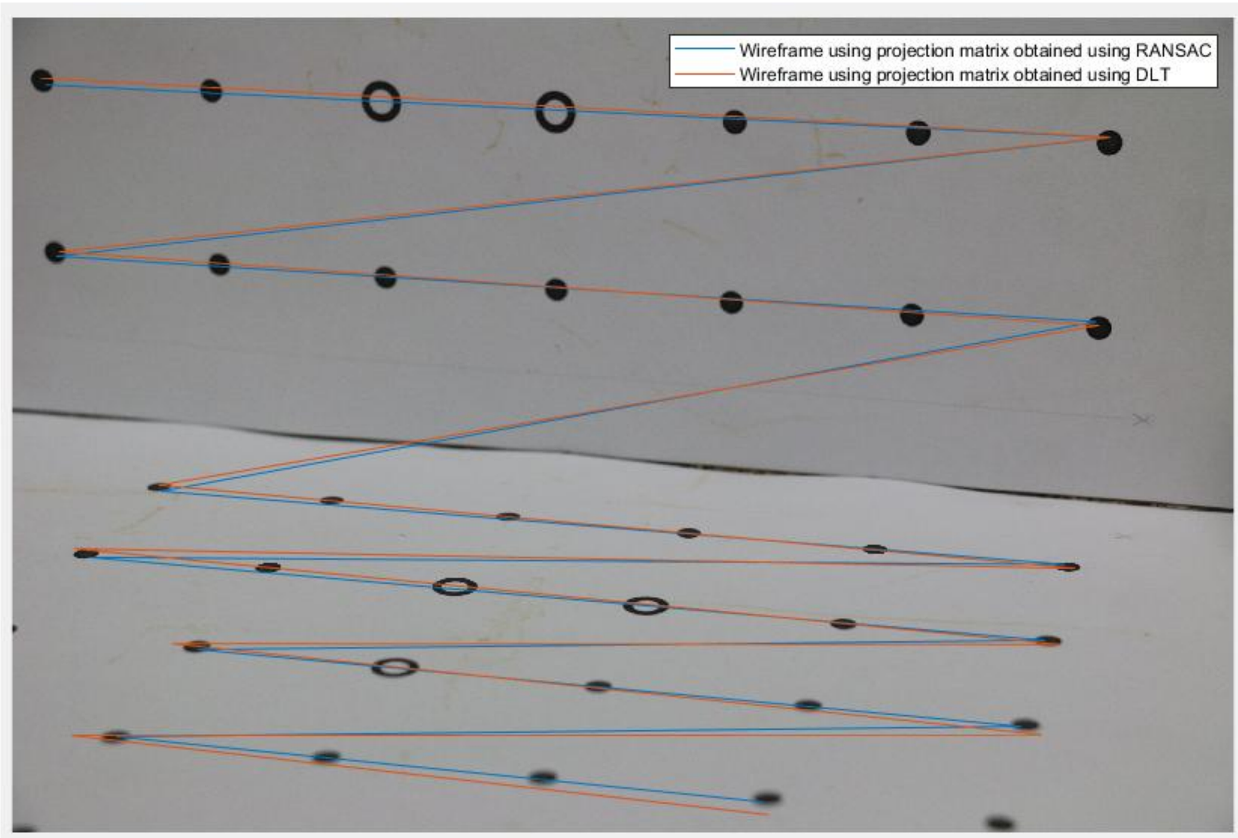
Wireframe using projection matrix obtained using DLT



Wireframe using projection matrix obtained using RANSAC DLT



The wireframe obtained using Ransac's Projection matrix is better than that of normal DLT's Projection matrix.



Code:

```
I = imread('IMG_5455.JPG');

% Wireframe obtained using P matrix obtained using DLT
genPts2D = (P * lut3D')';
genPts2D = genPts2D./genPts2D(:,3);

imshow(I); hold on; plot(a);
hold on;
plot(genPts2D(:,1), genPts2D(:,2));

% Wireframe obtained using P matrix obtained using DLT using Ransac
genRansacPts2D = (P_2 * lut3D')';
genRansacPts2D = genRansacPts2D./genRansacPts2D(:,3);

figure;
imshow(I); hold on; plot(a);
hold on;
plot(genRansacPts2D(:,1), genRansacPts2D(:,2));
```

Answer 6) Results:

Calibration Matrix :

```
K_2 =

   1.0e+04 *

    1.3488         0    0.2748
         0    1.3542    0.1836
         0         0    0.0001
```

Rotation Matrix :

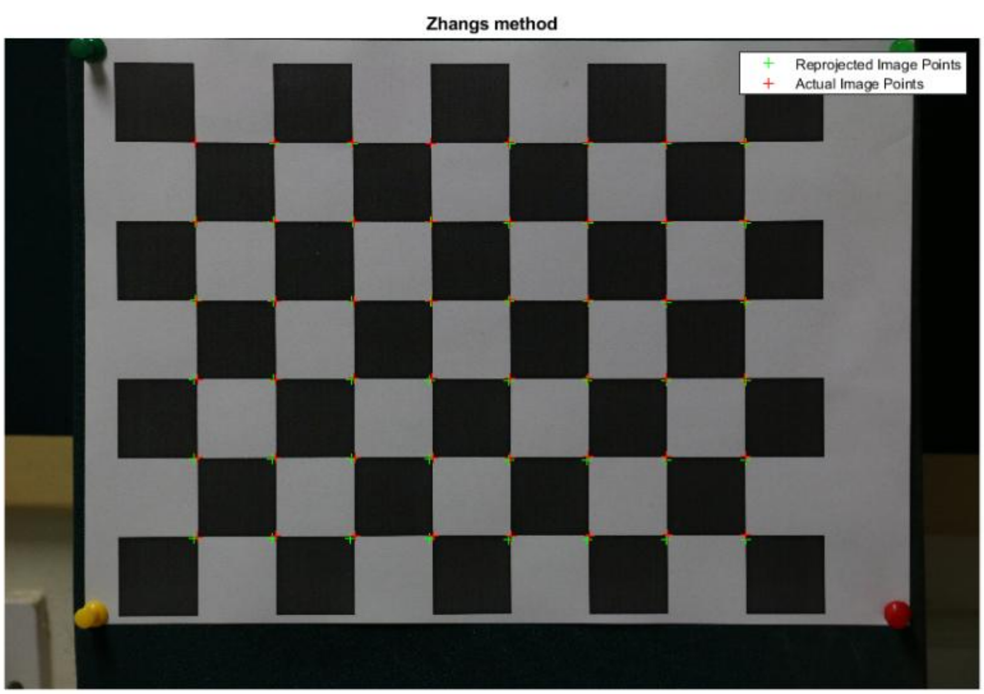
```
R_2 =

    0.9999   -0.0032    0.0130
    0.0030    0.9999    0.0167
   -0.0130   -0.0167    0.9998
```

Translation Vector :

```
t_2 =

 -110.3248
  -81.9840
  888.4892
```

Reprojection Error = 141(approx)

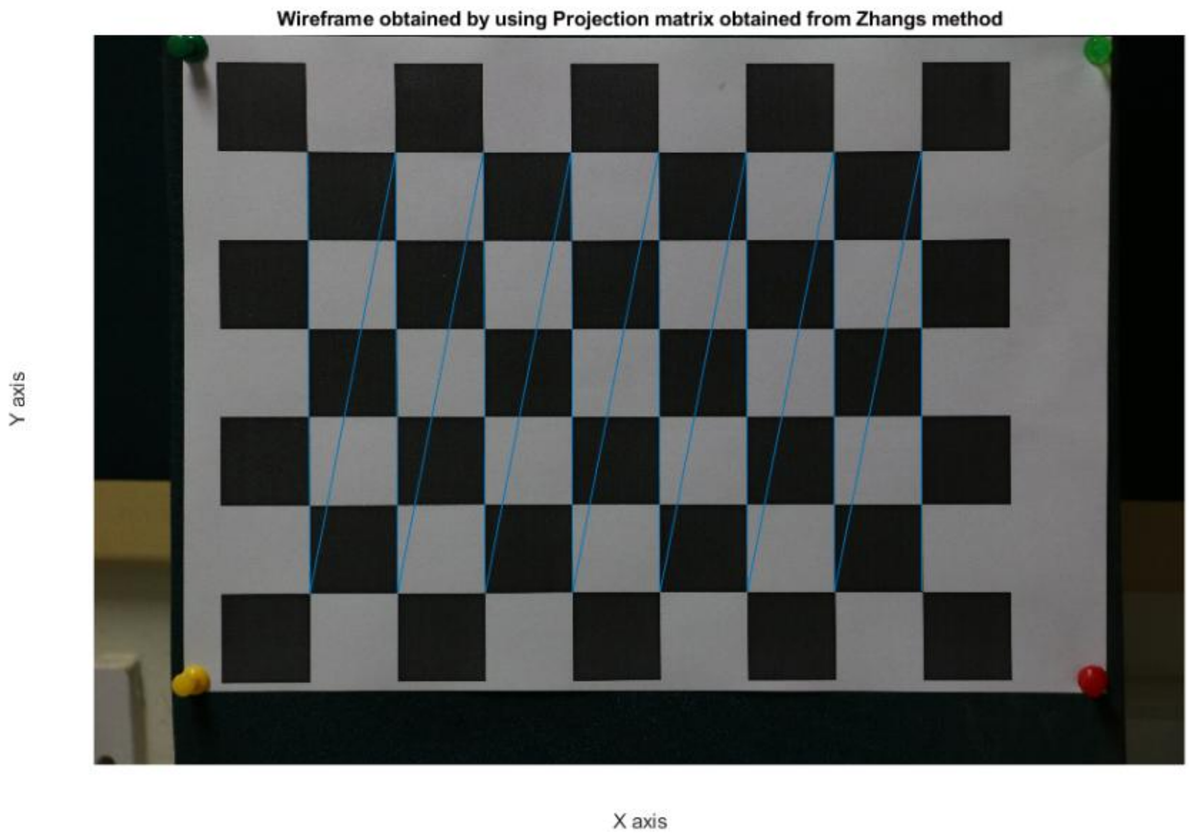
The error is considerably less than that of DLT method and DLT using Ransac method. Calibration using Zhang's method gives better camera parameters.



Code:

```
imagefiles = dir('Zhangs/*.JPG');
nfiles = length(imagefiles);    % Number of files found

images = {};
for index=1:nfiles
   currentfilename = imagefiles(index).name;
   images{index} = currentfilename;
end

[imagePoints,boardSize] = detectCheckerboardPoints(images);
squareSize = 29;
worldPoints = generateCheckerboardPoints(boardSize,squareSize);

I = imread(images{1});
imageSize = [size(I,1),size(I,2)];

cameraParams = estimateCameraParameters(imagePoints,worldPoints, ...
                                        'ImageSize',imageSize);

[imagePoints,boardSize] = detectCheckerboardPoints(I);
[rotationMatrix,translationVector] = extrinsics(...
    imagePoints,worldPoints,cameraParams);

Proj = cameraMatrix(cameraParams,rotationMatrix,translationVector);

homogeneousWorldPts = [worldPoints, zeros(length(worldPoints), 1),...
ones(length(worldPoints), 1)]';
homogeneousImgPts = Proj' * homogeneousWorldPts;
homogeneousImgPts = homogeneousImgPts./homogeneousImgPts(3,:);

% Show actual and reprojected image points
imshow(I); hold on;
plot(imagePoints(:,1), imagePoints(:,2), '+r');
plot(homogeneousImgPts(1,:), homogeneousImgPts(2,:), '+g');
title('Zhangs method');
legend('Actual Image Points', 'Reprojected Image Points');

% Reprojection Error
errZhang = (imagePoints - homogeneousImgPts(1:2,:)').^2;
errZhang = sum(sqrt(sum(errZhang(:,1:2), 2)));
```

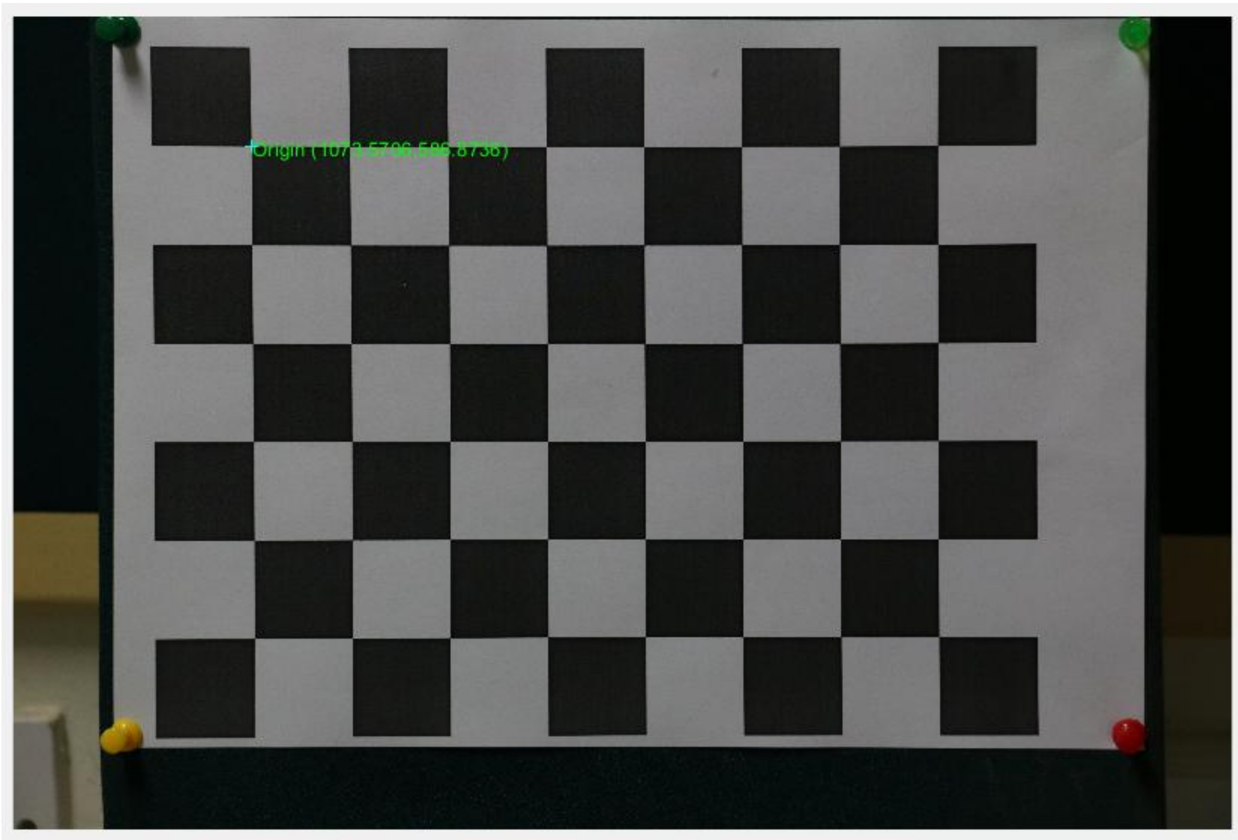
Answer 7) The wireframe obtained has straight lines which correspond with the straight lines of the checkerboard.



Code:

```
% Show wireframe
figure;
imshow(I); hold on;
plot(homogeneousImgPts(1,:), homogeneousImgPts(2,:));
title('Wireframe obtained by using P obtained from Zhangs method');
xlabel('X axis');
ylabel('Y axis');
```

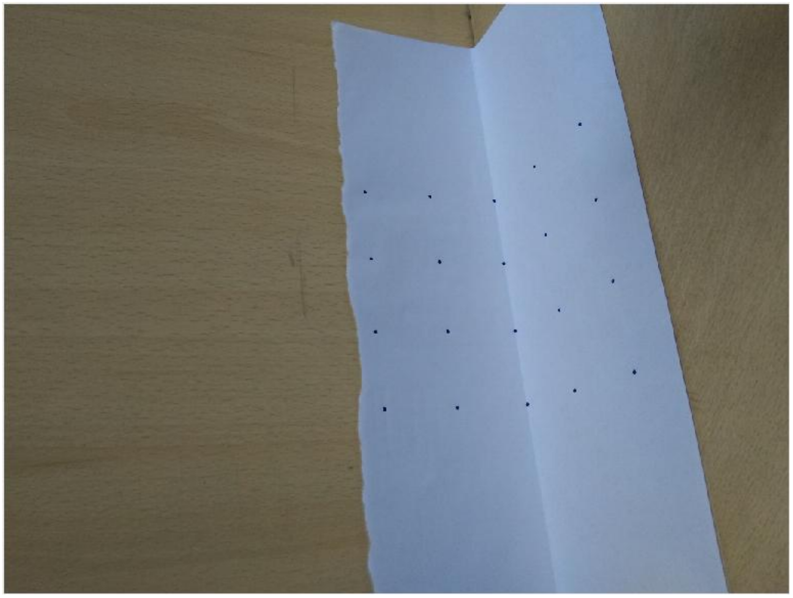
Answer 8) Error in the position of the actual and reprojected origin is 4 approximately. The error is so small because of which the two points seem coinciding only.



Code:

```
% Image of world origin
origin = [zeros(3,1); 1];
originImage = Proj' * origin;
originImage = originImage/originImage(3);
hold on;
plot(originImage(1), originImage(2), '+r');
text(originImage(1), originImage(2), ['Origin (',...
num2str(originImage(1)), ',', num2str(originImage(2)), ')'],...
    'Color', 'green');
```

Answer 10) Input images for DLT:



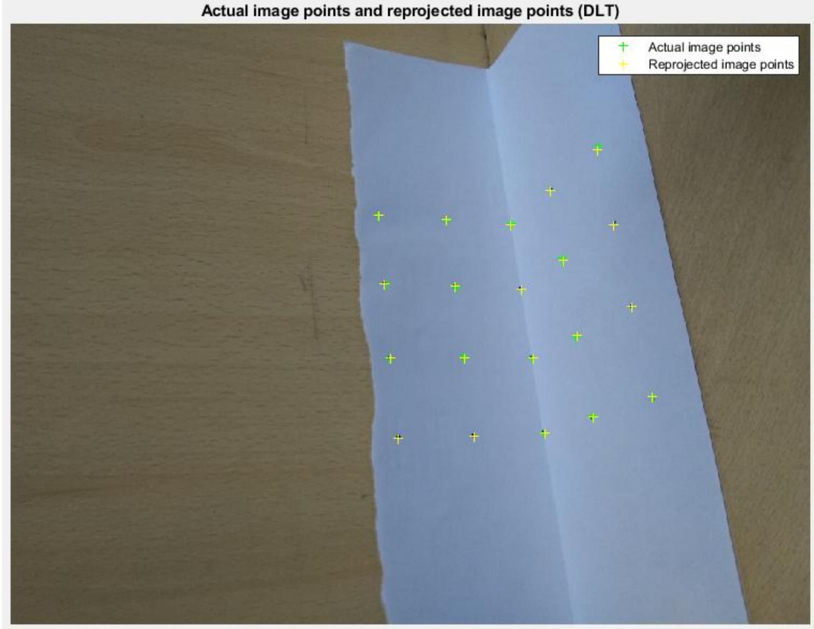
Results: The image when read in matlab got rotated by 90 degrees about Z axis due to which one of the focal lengths is negative.

Intrinsic and Extrinsic Parameters:

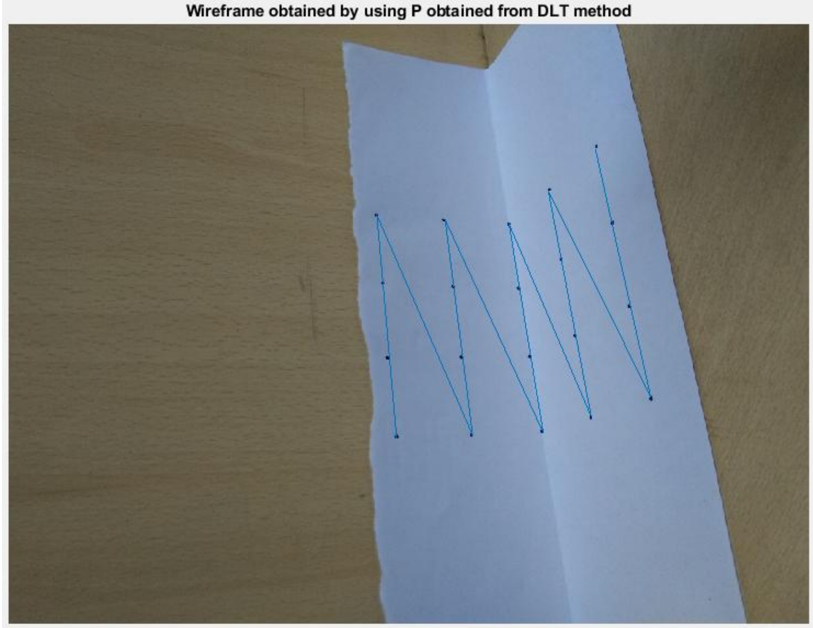
```
K_1 =

   1.0e+03 *

    4.1186    0.1019    2.0226
         0   -4.0356    1.5151
         0         0    0.0010

>> R_1

R_1 =

   -0.0917   -0.3147    0.9448
    0.9285   -0.3698   -0.0330
    0.3597    0.8742    0.3261

>> t_1

t_1 =

  -33.4526
  -14.4616
 -161.9863
```

Reprojection error = 8987.7 (approx) The error could be due to distortions present in the image.

Actual image points and reprojected image points (DLT)



Wireframe obtained by using P obtained from DLT method

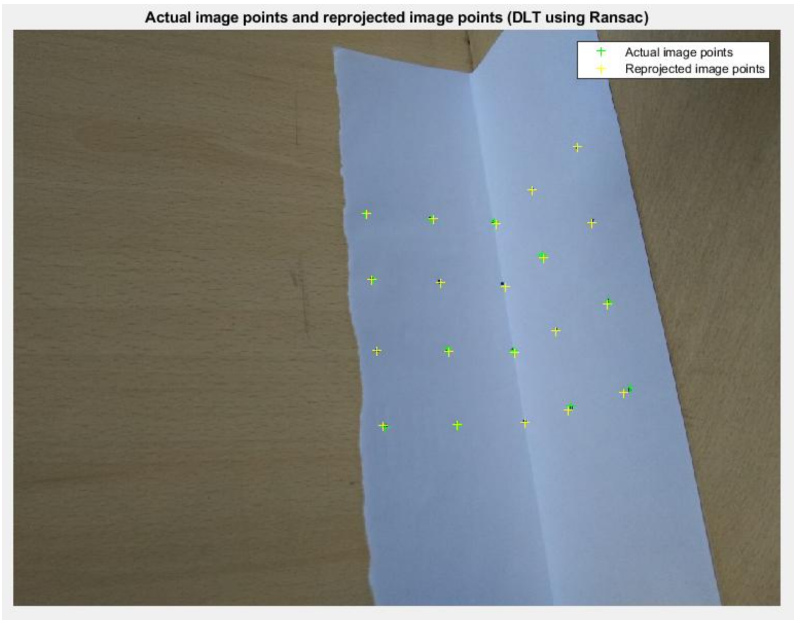


Result for DLT using Ransac:

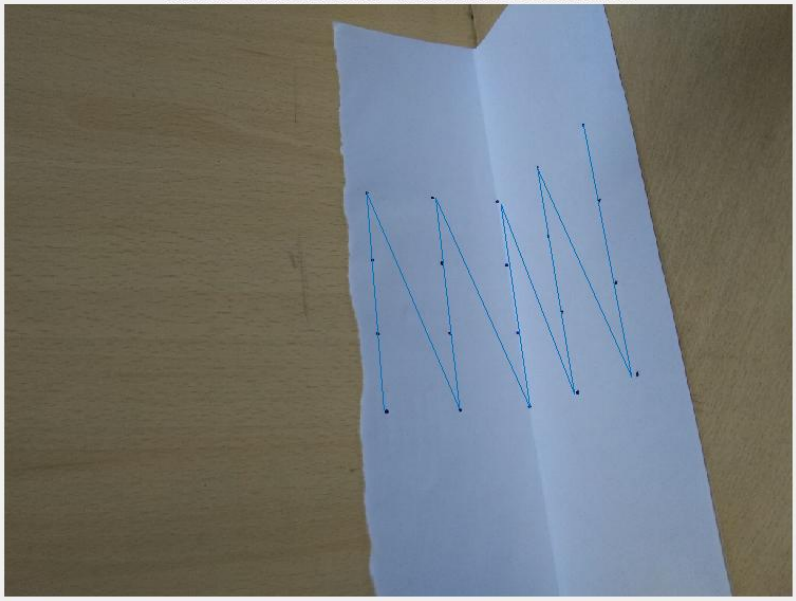
Intrinsics and extrinsics:

```
K_2 =

   1.0e+03 *

    3.7150   -0.0328    2.0985
         0   -3.6260    1.8963
         0         0    0.0010

>> R_2

R_2 =

   -0.0602   -0.2742    0.9598
    0.9608   -0.2767   -0.0188
    0.2708    0.9210    0.2801

>> t_2

t_2 =

  -31.7496
  -29.3034
 -145.4163
```

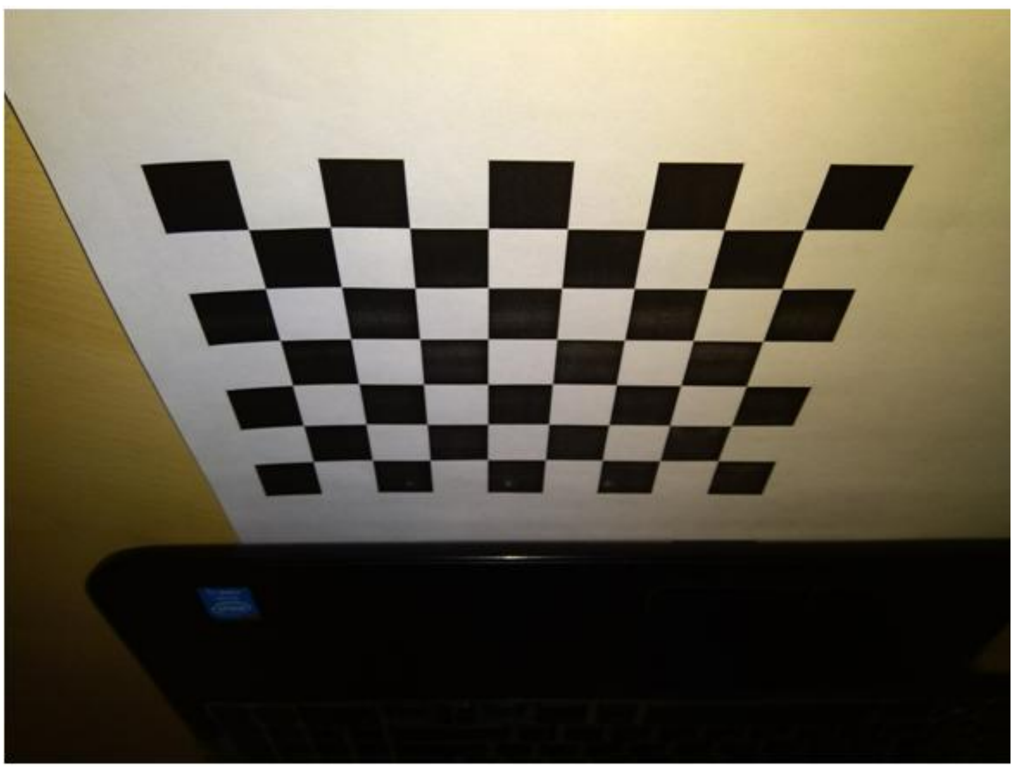
Reprojection error = 9003.2(approx)

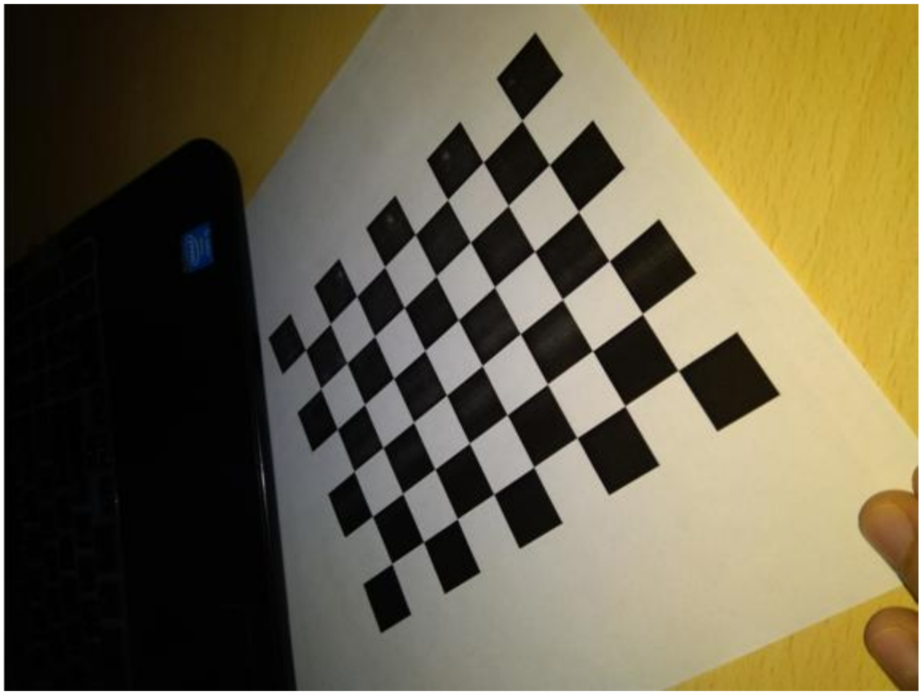


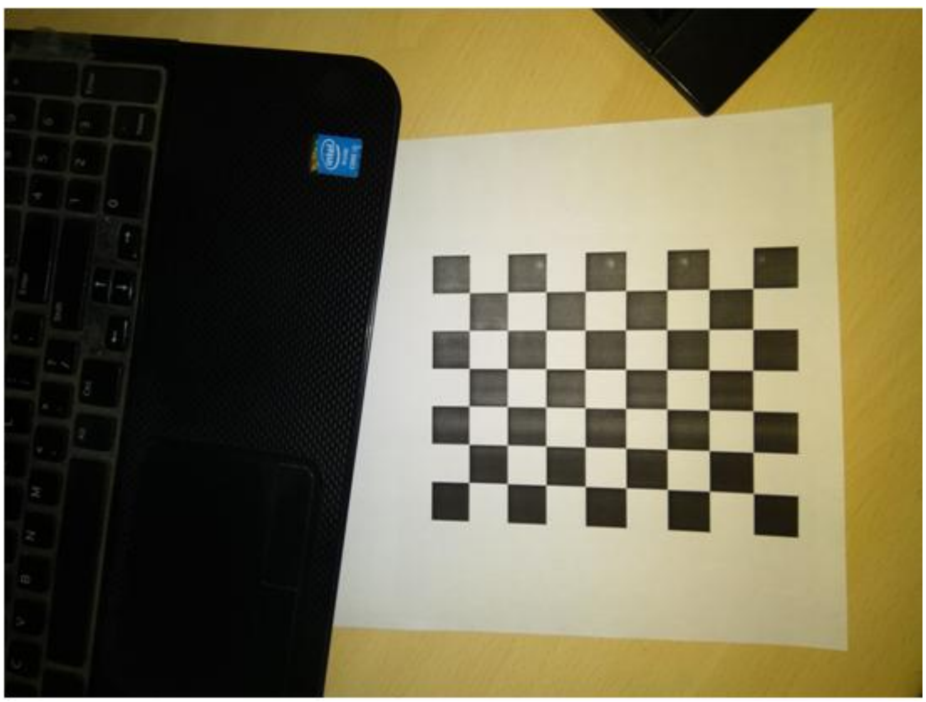
Wireframe obtained by using P obtained from DLT using Ransac



Input Images for Zhang:







Results:

Calibration Matrix:

```
K_3 =

   1.0e+03 *

    3.6893         0    2.2757
         0    3.6756    1.7134
         0         0    0.0010
```

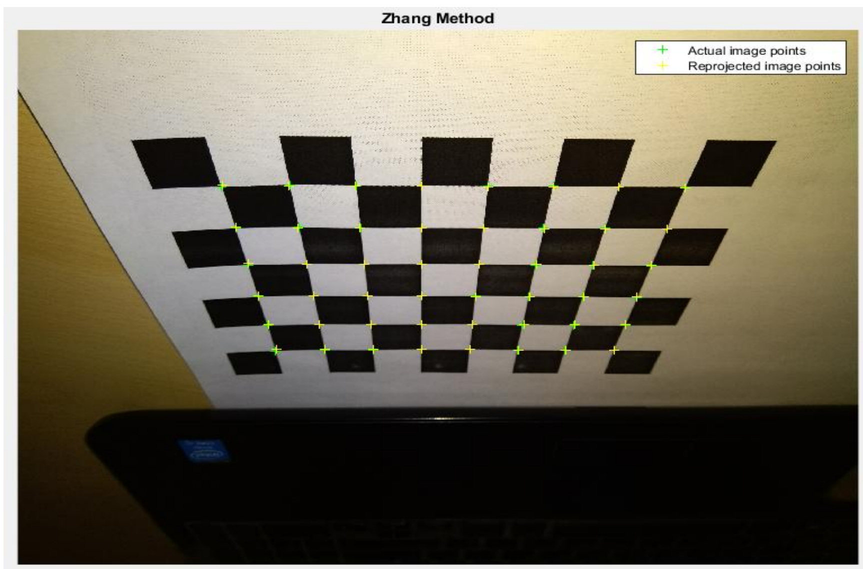
Rotation Matrix:

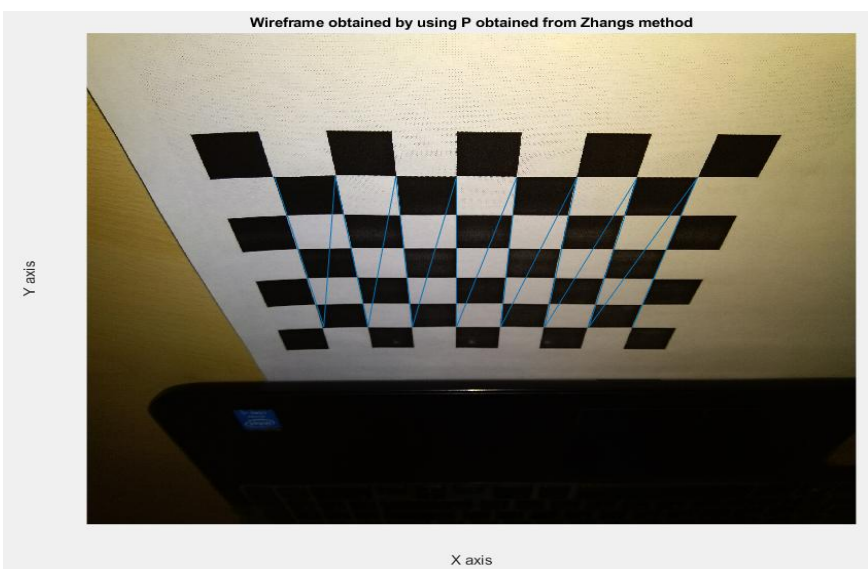
```
R_3 =

    1.0000   -0.0031    0.0066
    0.0030    0.9999    0.0101
   -0.0067   -0.0101    0.9999
```

Translation Vector:

```
t_3 =

 -110.9039
  -82.2264
  893.4950
```

Reprojection Error = 222 (approx) The error is much higher compared to the one done in answer 6. This could be due to the poor quality of the checkerboard printout which resulted in poor image of the checkerboard.





Code:

```
I = imread('MyCamDLT/1.jpg');
I_g = rgb2gray(I);
I_b = I_g;
I_b(I_g > 50) = 255;
I_b(I_g <= 50) = 0;
I_m = imerode(I_b, ones(2,2));


corners = detectHarrisFeatures(I_m);
a = corners.selectStrongest(25);

lut2D = [3386, 703, 1; 3473, 1156, 1; 3576, 1627, 1; 3701,2146, 1;...
         3117, 9525, 1; 3179, 1347, 1; 3260, 1806, 1; 3353, 2256, 1;...
         2887, 1154, 1; 3180, 1363, 1; 3003, 1925, 1; 3074, 2357, 1;...
         2510, 1136, 1; 2568, 1518, 1; 2610, 1915, 1; 2672, 2374, 1;...
         2119, 1102, 1;2164, 1497, 1; 2185, 1920, 1; 2236, 2388, 1];
lut3D = [0, 32, 0, 1; 16, 32, 0, 1; 32, 32, 0, 1; 48, 32, 0, 1;...
         0, 16, 0, 1; 16, 16, 0, 1; 32, 16, 0, 1; 48, 16, 0, 1;...
         0, 0, 0, 1; 16, 0, 0, 1; 32, 0, 0, 1; 48, 0, 0, 1;...
         0, 0, 16, 1; 16, 0, 16, 1; 32, 0, 16, 1; 48, 0, 16, 1;...
         0, 0, 32, 1; 16, 0, 32, 1; 32, 0, 32, 1; 48, 0, 32, 1];


pts_2D = lut2D([1,15,7,12,20,17],:);
pts_3D = lut3D([1,15,7,12,20,17],:);

P = calcProjMatrix(pts_2D, pts_3D);
[K_1, R_1, t_1] = decompose_P(P);

close all; imshow(I); hold on; plot(a);
genPts2D = genReprojPts(P, lut3D);
hold on;
plot(genPts2D(:,1), genPts2D(:,2), '+y');
legend('Actual image points', 'Reprojected image points');
title('Actual image points and reprojected image points (DLT)');
% Reprojection Error
err = (lut2D - genPts2D).^2;
err = sum(sqrt(sum(err(:,1:2), 2)));


% Wireframe
close all;
imshow(I);
hold on;
plot(genPts2D(:,1), genPts2D(:,2));
title('Wireframe obtained by using P obtained from DLT method');
xlabel('X axis');
ylabel('Y axis');
```

```matlab
% DLT using Ransac
[P_2, K_2, R_2, t_2] = calibrateCameraRansac(lut2D, lut3D);

close all;
figure;imshow(I); hold on; plot(a);
genRansacPts2D = genReprojPts(P_2, lut3D);
hold on; plot(genRansacPts2D(:,1), genRansacPts2D(:,2), '+y');
legend('Actual image points', 'Reprojected image points');
title('Actual image and reprojected image points (DLT using Ransac)');
% Reprojection error
errRansac = (lut2D - genRansacPts2D).^2;
errRansac = sum(sqrt(sum(errRansac(:,1:2), 2)));
% Wireframe
close all; imshow(I); hold on;
plot(genRansacPts2D(:,1), genRansacPts2D(:,2));
title('Wireframe obtained by using P obtained from DLT using Ransac');
xlabel('X axis'); ylabel('Y axis');

% Applying Zhangs method
imagefiles = dir('MyCamZhang/*.JPG');
nfiles = length(imagefiles);    % Number of files found
images = {};
for index=1:nfiles
   currentfilename = imagefiles(index).name;
   images{index} = currentfilename;
end
[imagePoints,boardSize] = detectCheckerboardPoints(images);
squareSize = 16;
worldPoints = generateCheckerboardPoints(boardSize,squareSize);
I = imread(images{1});
imageSize = [size(I,1),size(I,2)];
cameraParams = estimateCameraParameters(imagePoints,worldPoints, ...
                                        'ImageSize',imageSize);
[imagePoints,boardSize] = detectCheckerboardPoints(I);
[rotationMatrix,translationVector] = extrinsics(...
    imagePoints,worldPoints,cameraParams);
Proj = cameraMatrix(cameraParams,rotationMatrix,translationVector);
homogeneousWorldPts = [worldPoints, zeros(length(worldPoints), 1),...
    ones(length(worldPoints), 1)]';
homogeneousImgPts = Proj' * homogeneousWorldPts;
homogeneousImgPts = homogeneousImgPts./homogeneousImgPts(3,:);

figure; imshow(I); hold on; plot(imagePoints(:,1), imagePoints(:,2), '+g');
hold on; plot(homogeneousImgPts(1,:), homogeneousImgPts(2,:), '+y');
title('Zhang Method');
legend('Actual image points', 'Reprojected image points');
% Reprojection Error
errZhang = (imagePoints - homogeneousImgPts(1:2,:)').^2;
errZhang = sum(sqrt(sum(errZhang(:,1:2), 2)));
% Wireframe
imshow(I); hold on;
plot(homogeneousImgPts(1,:), homogeneousImgPts(2,:));
title('Wireframe obtained by using P obtained from Zhangs method');
xlabel('X axis'); ylabel('Y axis');
```