ECE 661 (Fall 2016) - Computer Vision - HW 9

Debasmit Das

November 22, 2016

1 Objective

The goal of this homework is to estimate the intrinsic and extrinsic camera parameters using Zhang's calibration algorithm.

2 Corner Detection

For corner detection the following steps are carried out -

- The Canny edge detector is applied on the calibration pattern to find edges. The threshold of 0.7 is used.
- Hough Transform was applied to construct hough lines such that the gap between unconnected lines are filled.
- The intersection of these lines are the corner points. So these corner points are numbered accordingly. The order was from top to bottom and from left to right.

3 Calibration

After the corners were detected, the next step is to find the homography between world coordinates and image pixels. The corner coordinates are measured using the ruler where the top left corner has coordinates of (0,0). H is found such that where $x_I = Hx_W$ for each image. H was found using nullspace of matrix as in previous homeworks. The solution for nullspace was found to be an eigenvector that corresponded to the smallest eigenvalues after using SVD.

After the homographies are found the Zhang's algorithm is applied to find the intrinsic parameter $K \in \mathbb{R}^{3\times 3}$ and extrinsic parameters R, t. The assumption of the algorithm is that the calibration pattern is on $Z = 0$ and pictures of calibration pattern are taken from different viewpoints. It is based on the assumption that the image of the absolute conic are independent of the viewpoint or relative pose of the camera. The image of the absolute conic is given by $w = K^{-T}K^{-1}$.
We can get $h_1^T wh_1 = h_2^T wh_2$ and $h_1^T wh_2 = 0$ where w is the image of the absolute conic and vectors h_i are columns of homography matrix H. We can reshape matrix w into a vector $b = (w_{11}, w_{12}, w_{22}, w_{13}, w_{23}, w_{33})$ and define $v_{ij} = (h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2}, h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3})^T$. We can derive $v_{12}^T b = 0$ and $(v_{11} - v_{22})^T b = 0$. Since v would consists only values of matrix H, we can stack the result using n instances in the data set. b is the null-space solution and can be found using SVD as the eigen-vector corresponding to the smallest eigenvalue.
Even though the image of the conic is found out, we still have to find the parameters of the

camera. The camera calibration matrix is found as the following $K = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$. The parameters can be found using the following equations:

$$x_0 = \frac{w_{12} - w_{13} - w_{11}w_{23}}{w_{11}w_{22} - w_{12}^2} \tag{1}$$

$$x_0 = w_{33} - \frac{w_{13}^2 + x_0(w_{12}w_{13} - w_{11}w_{23})}{w_{11}} \tag{2}$$

$$\alpha_x = \sqrt{\frac{\lambda}{w_{11}}} \tag{3}$$

$$\alpha_y = \sqrt{\frac{\lambda w_{11}}{w_{11}w_{22} - w_{12}^2}} \tag{4}$$

$$s = -\frac{w_{12}\alpha_x^2\alpha_y}{\lambda} \tag{5}$$

$$y_0 = \frac{sx_0}{\alpha_y} - \frac{w_{13}\alpha_x^2}{\lambda} \tag{6}$$

The intrinsic parameters would be the same for all images but we have to calculate the extrinsic parameter separately for each image depending on the viewpoint. They are calculated for each image separately.
$R = [r_1, r_2, r_3]$ and $K^{-1}[h_1, h_2, h_3] = [r_1, r_2, t]$ From these, we get
$r_1 = \epsilon K^{-1}h_1$
$r_2 = \epsilon K^{-1}h_2$
$r_3 = r_1 \times r_2$
$t = \epsilon K^{-1}h_3$,
where $\epsilon = \frac{1}{\|K^{-1}h_1\|}$

4 Refining Calibration Parameters

Since, corner detection might not give exact location of corners we use Levenberg-Marquadt (LM) algorithm to refine the results. The Matlab's inbuilt function is used for LM. The cost function for the optimization is as follows $\texttt{Cost} = \sum_i \sum_j \left\| x_{ij} - K[R_i|t_i]x_{M,j} \right\|^2$, where x_{ij} is the j^{th} pixel of the image i that was estimated using corner detection algorithm, $x_{M,j}$ is the world coordinate j, R_i,t_i are extrinsic parameters from image i, and K is intrinsic parameters of the camera. All the parameters of the camera are stacked into $p = [K, R_1, t_1, R_2...]$ vector and use LM algorithm to refine these parameters.
Since the matrix R has 9 parameters, but the rotation has only 3 DoF. So we use polar coordinates and transform matrix R into Rodriguez representation. $w = \frac{\phi}{2sin\phi}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$ and $\phi = cos^{-1}\frac{trace(R)-1}{2}$. To transform back from Rodriguez, we use $R = I + \frac{sin\phi}{\phi}[W]_x + \frac{1-cos\phi}{\phi}[W]_x^2$, where $W_x = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}$.

5 Conditioning Rotation Matrix

In every step, we need to make sure that the rotation matrix R is orthonormal. We need to find the best orthonormal match to the given matrix Q. we have to minimize $\|R - Q\|_F^2$, where $\|.\|_F$ is the Frobenius norm. The solution to the problem is found by carrying out SVD. $[U, D, V] = SVD(Q)$ and we set R equal to UV^T.

6 Incorporating Radial Distortion

We have assumed to have a pin-hole camera model. But the pin-hole camera model breaks down for shorter focal-length. A radial distortion is introduced for which we need to compensate for. The projected pixel $\hat{x} = K[R|t]x_W$ need to be modified to take care of the radial distortion.

$$\hat{x}_{rad} = \hat{x} + (\hat{x} - x_0)[k_1 r^2 + k_2 r^4] \tag{7}$$

$$\hat{y}_{rad} = \hat{y} + (\hat{y} - y_0)[k_1 r^2 + k_2 r^4] \tag{8}$$

where k_1, k_2 are parameters for radial distortion and $r^2 = (x - \hat{x}_0)^2 + (y - \hat{y}_0)^2$.

7 Results

For Provided Dataset

Intrinsic parameters

Intrinsic parameters before LM

$$K = \begin{bmatrix} 728.827 & 1.563 & 318.89 \\ 0 & 727.18 & 238.28 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Intrinsic parameters after LM, without radial distortion

$$K = \begin{bmatrix} 943.53 & 1.77 & 319.8 \\ 0 & 942.89 & 235.3 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

Intrinsic parameters after LM, with radial distortion

$$K = \begin{bmatrix} 944.01 & 1.6889 & 318.42 \\ 0 & 944.1279 & 232.16 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

$k_1 = -0.2054$, $k_2 = 1.0150$

Output Images for detecting Corners

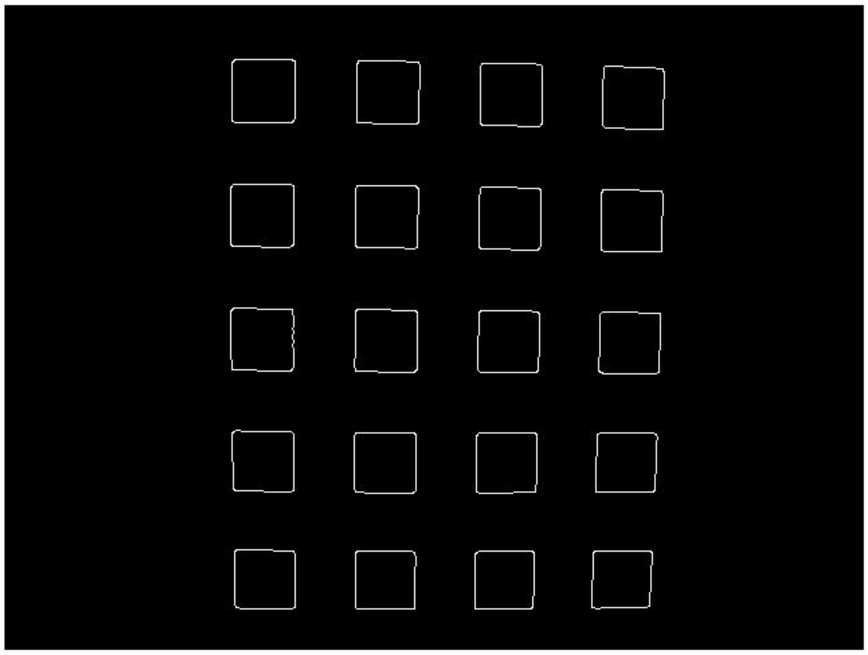

Figure 1: Canny Edge Detection Output for Pic 28 of Dataset

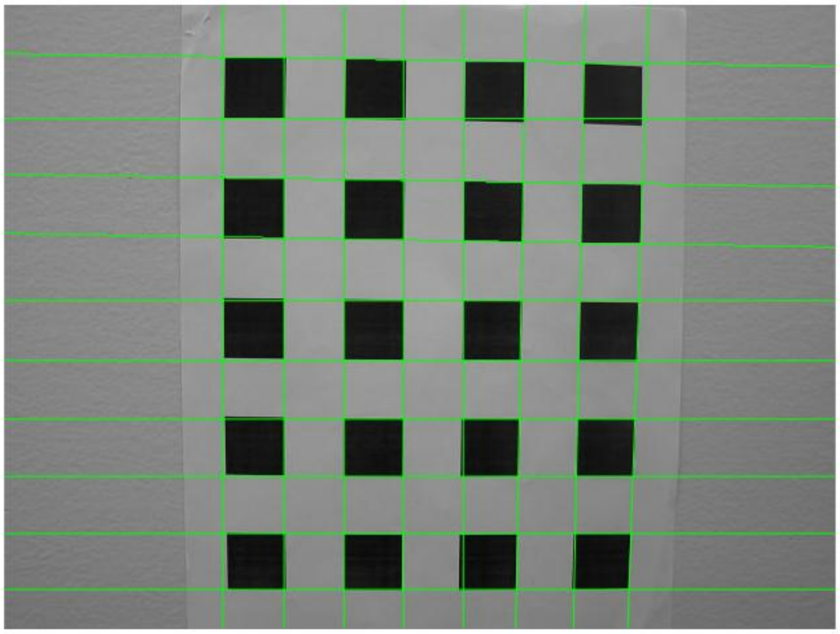

Figure 2: Hough Lines for Pic 28 of Dataset

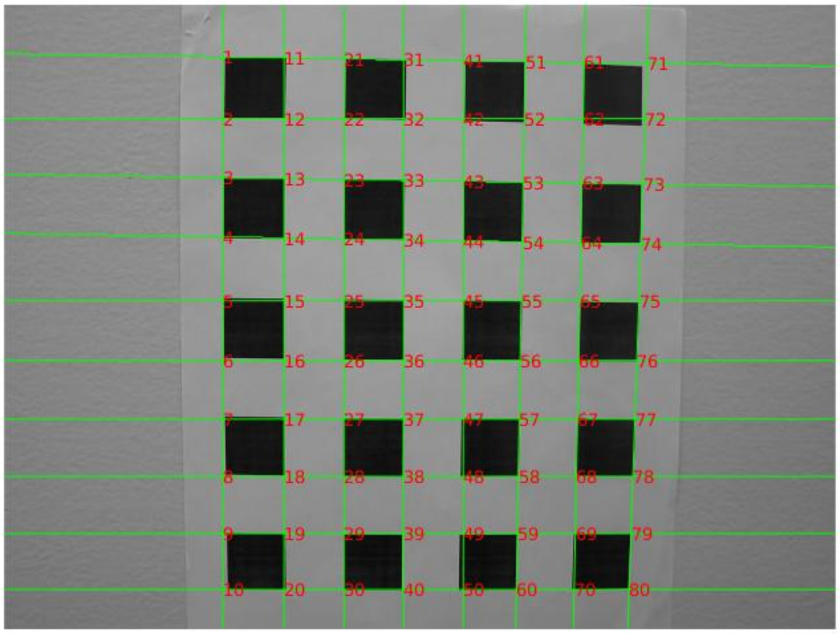

Figure 3: Corners for Pic 28 of Dataset

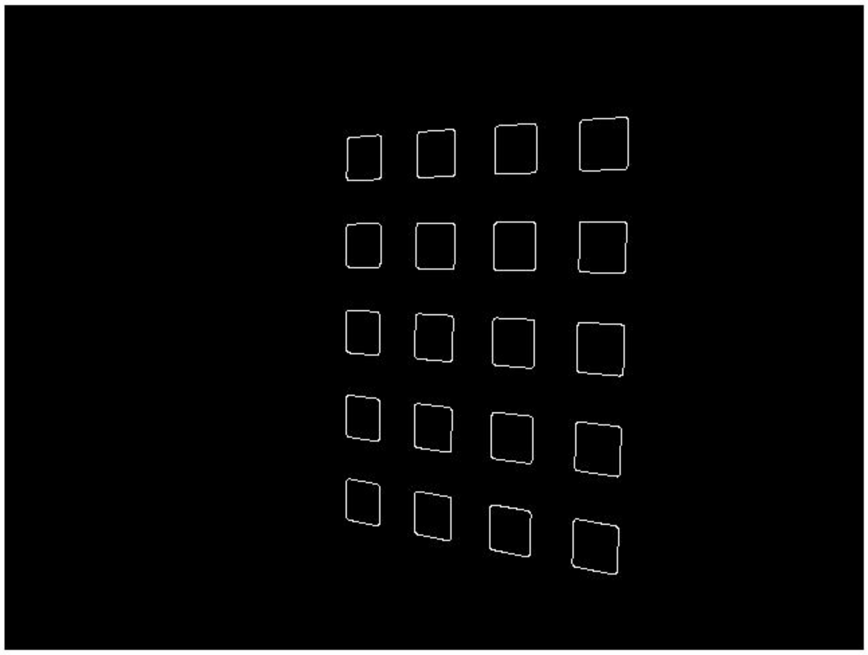

Figure 4: Canny Edge Detection Output for Pic 40 of Dataset

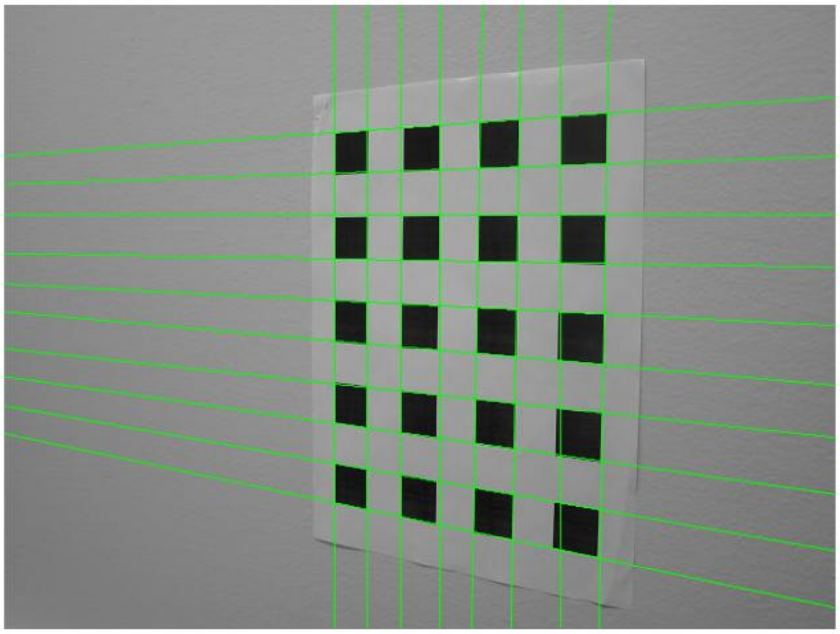

Figure 5: Hough Lines for Pic 40 of Dataset

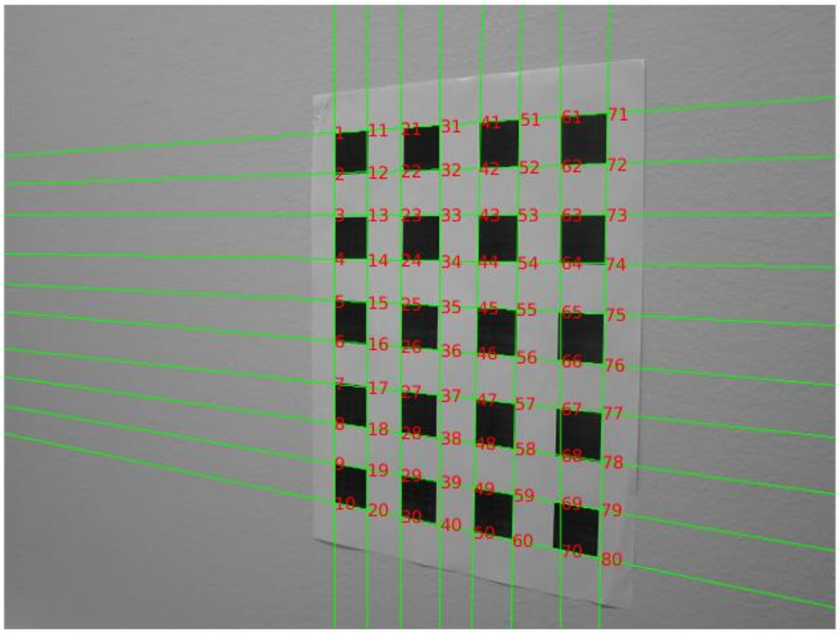

Figure 6: Corners for Pic 40 of Dataset

Images for extrinsic parameters

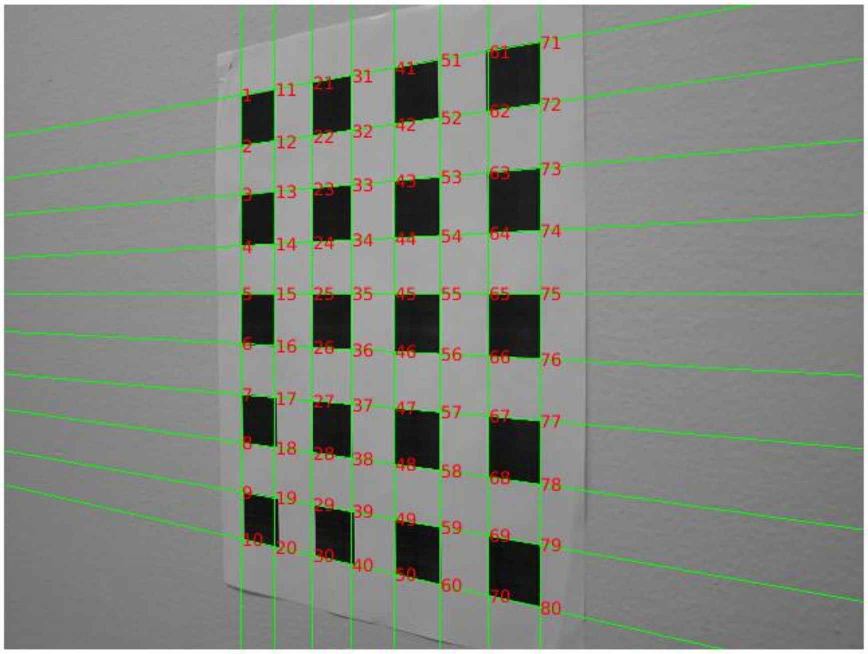

Figure 7: Extrinsic Parameters for Pic 20 of dataset

$$[R|t] = \begin{bmatrix} 0.8247 & -0.0048 & 0.6539 & -50.2311 \\ 0.0483 & 0.997 & -0.0742 & -105.9935 \\ -0.6522 & 0.0885 & 0.8223 & 724.1905 \end{bmatrix}$$

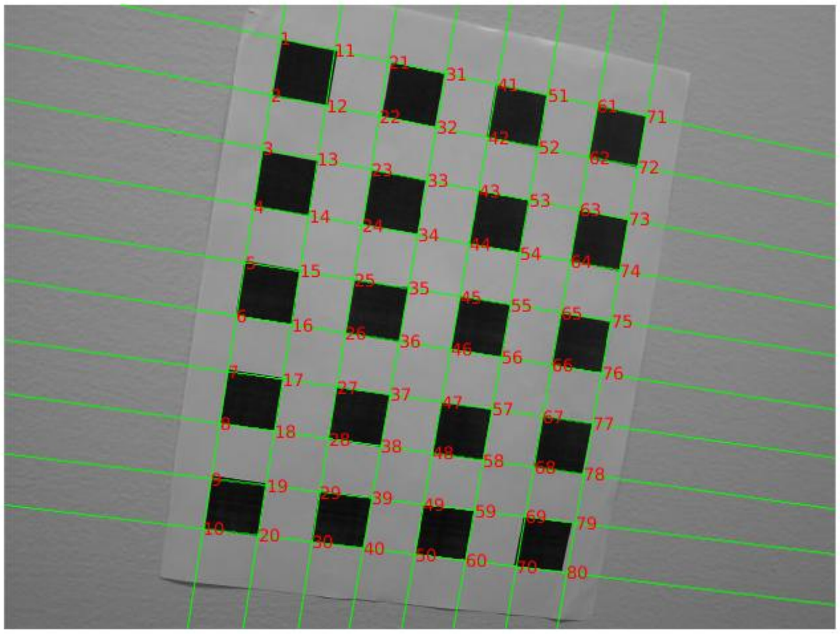

Figure 8: Extrinsic Parameters for Pic 23 of dataset

$$[R|t] = \begin{bmatrix} 0.9876 & -0.1681 & -0.2285 & -62.511 \\ 0.1618 & 0.9946 & -0.0948 & -121.9222 \\ 0.2331 & 0.0830 & 0.9906 & 549.3707 \end{bmatrix}$$

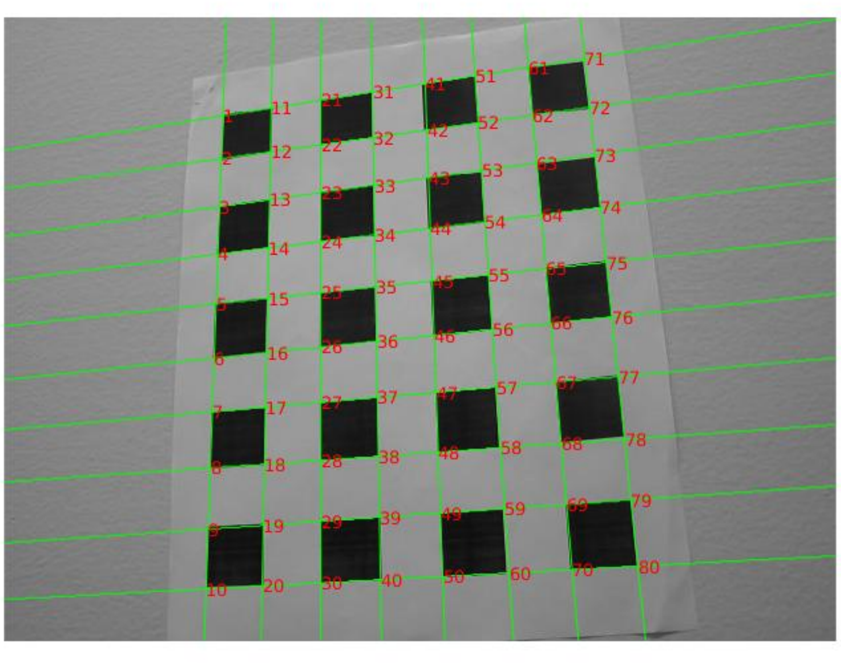

Figure 9: Extrinsic Parameters for Pic 27 of dataset

$$[R|t] = \begin{bmatrix} 0.9781 & 0.0332 & 0.2752 & -95.9517 \\ -0.1002 & 0.9443 & 0.4279 & -102.5573 \\ -0.2584 & -0.4382 & 0.9250 & 604.1141 \end{bmatrix}$$

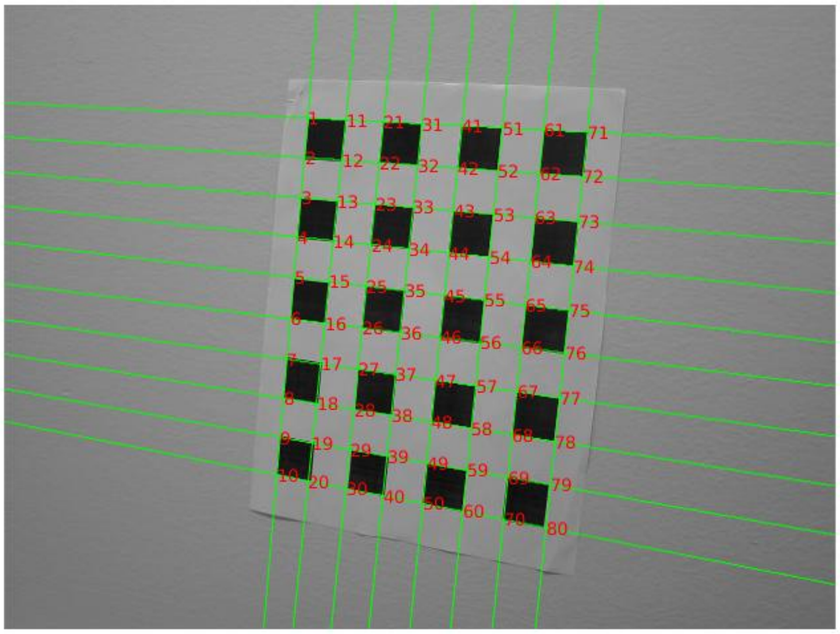

Figure 10: Extrinsic Parameters for Pic 38 of dataset

$$[R|t] = \begin{bmatrix} 0.9360 & -0.1031 & 0.4744 & -68.63 \\ 0.1134 & 0.9964 & -0.0263 & -119.7376 \\ -0.4721 & 0.0541 & 0.9387 & 767.2623 \end{bmatrix}$$

Output Images for Re-projection and also showing benefit of LM

We select Pic 28 of the dataset as the 'Fixed image'
Green markers are the ground truth and red markers are the projected ones

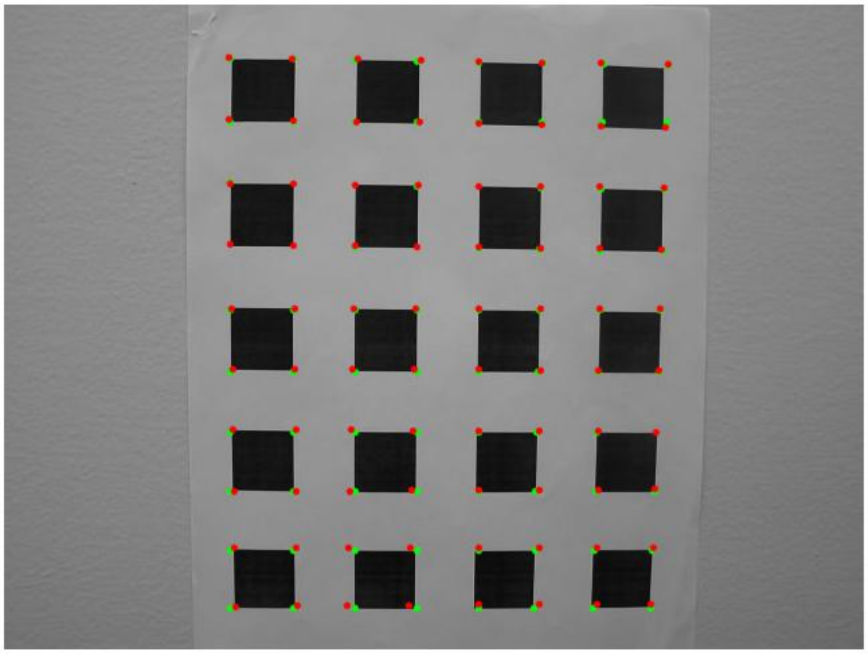

Figure 11: Re-projecting Pic 21 to 28 before LM (Error Mean = 1.372, Error Variance = 1.343)

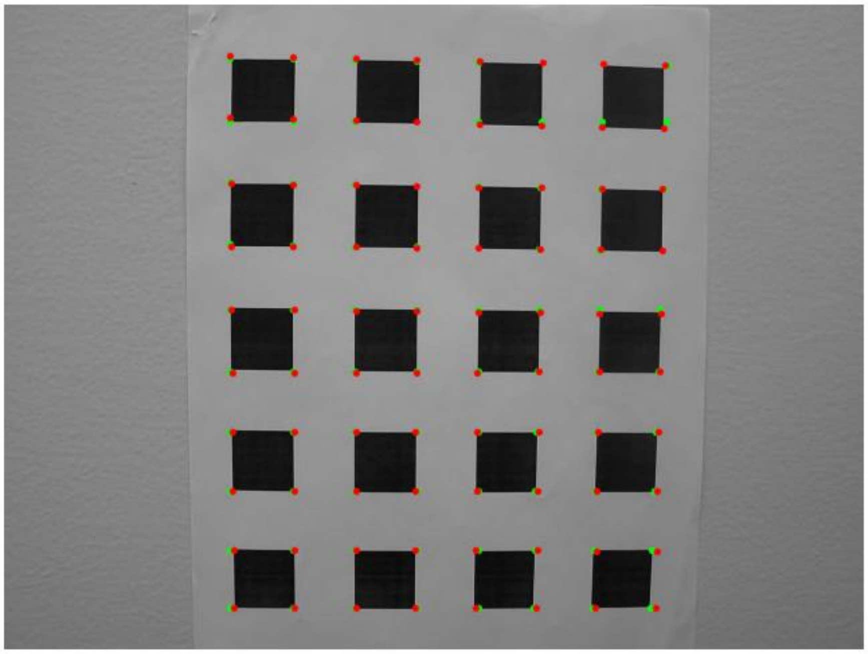

Figure 12: Re-projecting Pic 21 to 28 after LM(Error Mean = 0.8854, Error Variance = 0.8161)

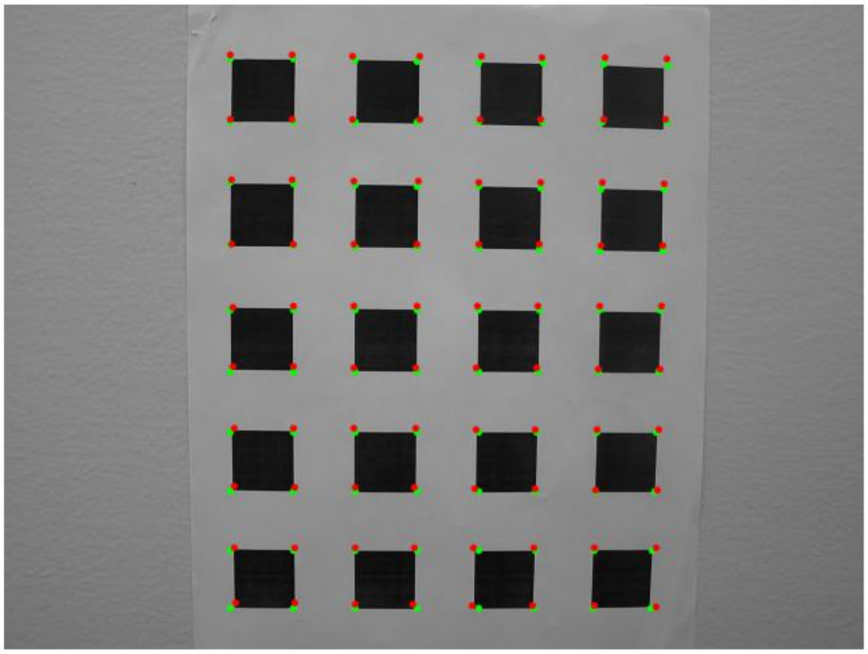

Figure 13: Re-projecting Pic 23 to 28 before LM (Error Mean = 1.9952, Error Variance = 1.5766)

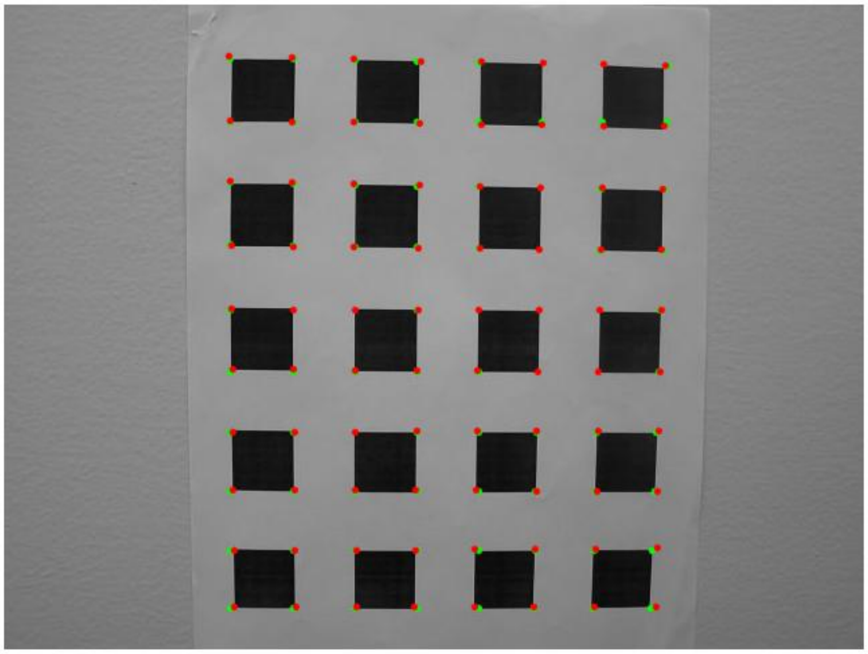

Figure 14: Re-projecting Pic 23 to 28 after LM (Error Mean = 0.9454, Error Variance = 0.6729)

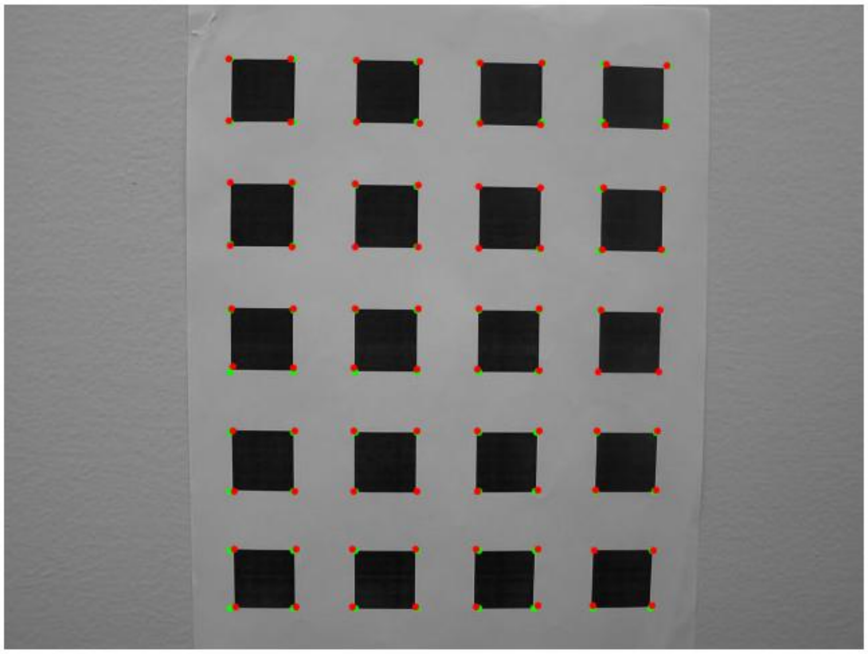

Figure 15: Re-projecting Pic 26 to 28 before LM (Error Mean = 1.0139, Error Variance = 0.5687)

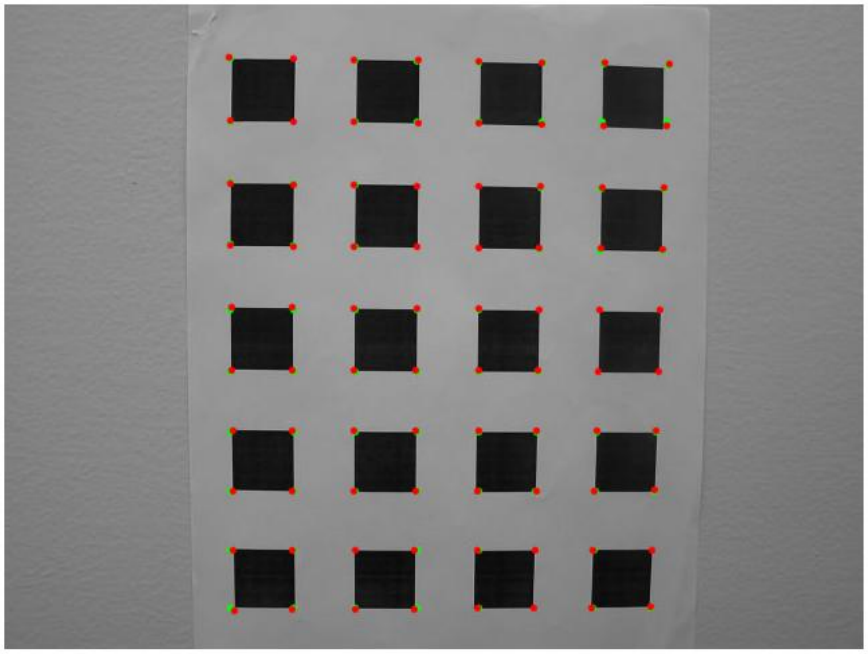

Figure 16: Re-projecting Pic 26 to 28 after LM (Error Mean = 0.6711, Error Variance = 0.3209)

For Created Dataset

Intrinsic parameters

Intrinsic parameters before LM

$$K = \begin{bmatrix} 1218.8 & -0.2 & 366.6 \\ 0 & 1215.4 & 682.5 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

Intrinsic parameters after LM, without radial distortion

$$K = \begin{bmatrix} 1545.5 & -6.4 & 368.3 \\ 0 & 1542.2 & 670.2 \\ 0 & 0 & 1 \end{bmatrix} \tag{13}$$

Intrinsic parameters after LM, with radial distortion

$$K = \begin{bmatrix} 1524.7 & -3.9 & 359.4 \\ 0 & 1525.9 & 624.2 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$k_1 = 0.2549$, $k_2 = -0.9438$

Output Images for detecting Corners

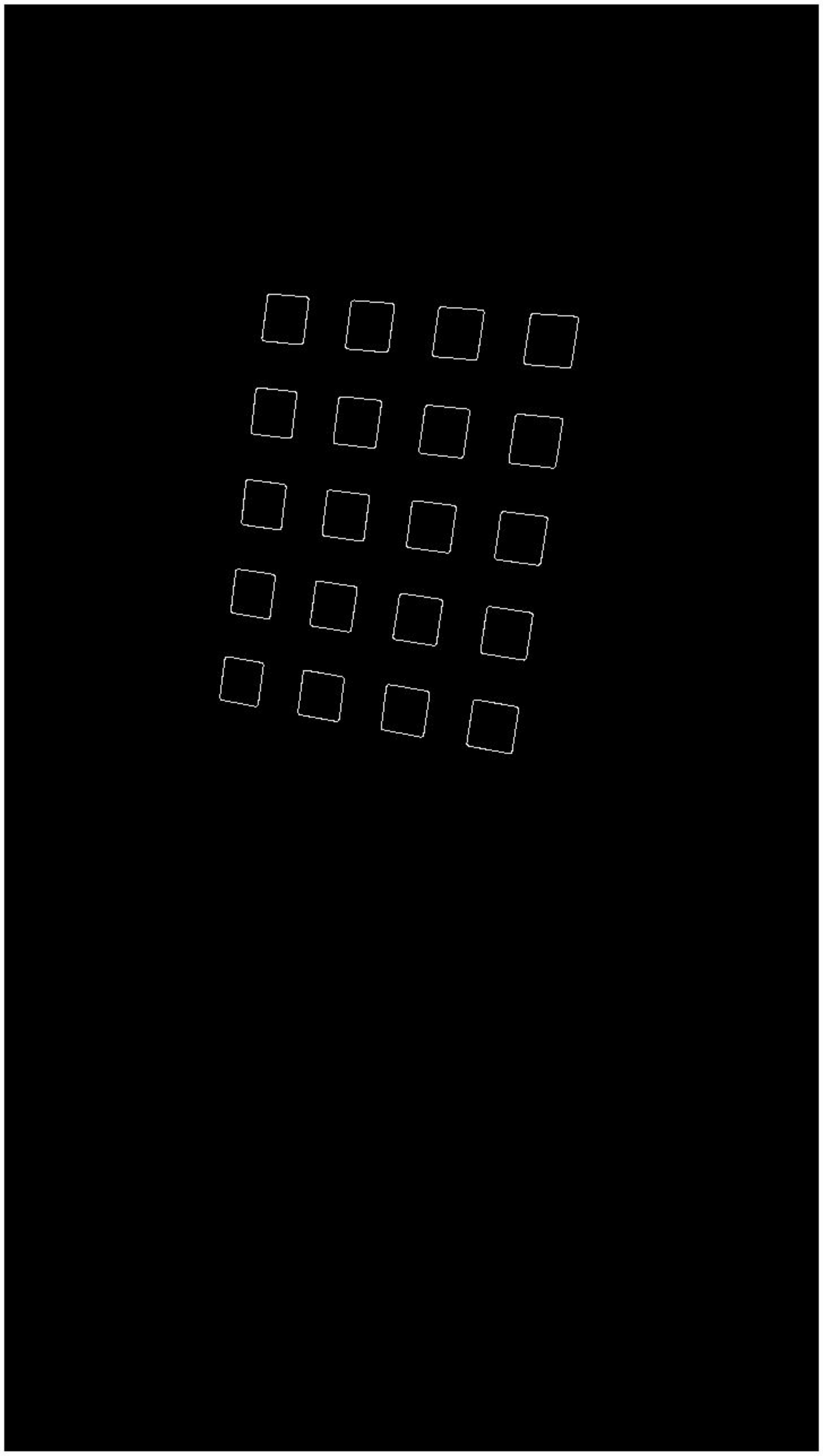

Figure 17: Canny Edge Detection Output for Pic 1 of Dataset

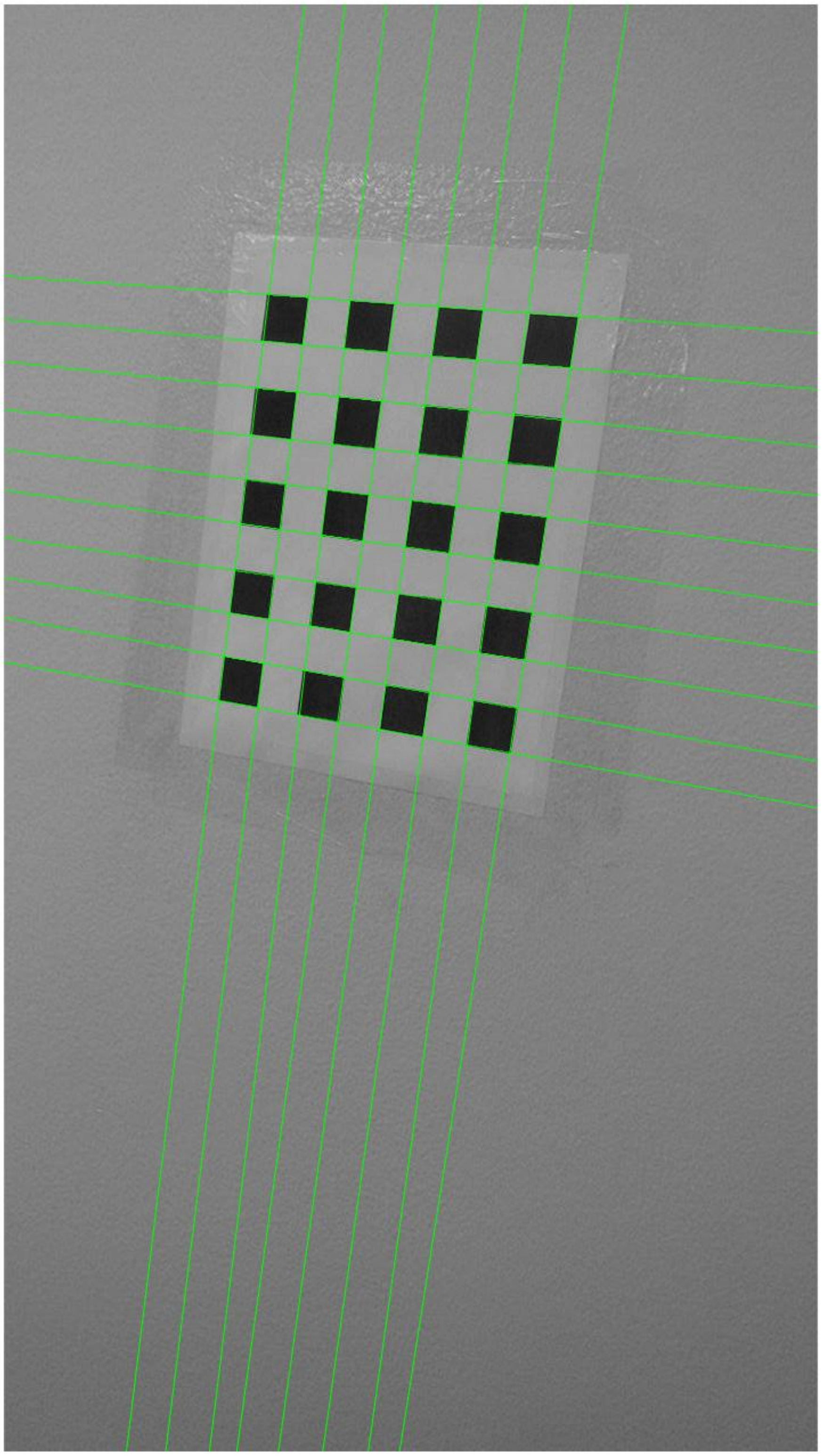

Figure 18: Hough Lines for Pic 1 of Dataset

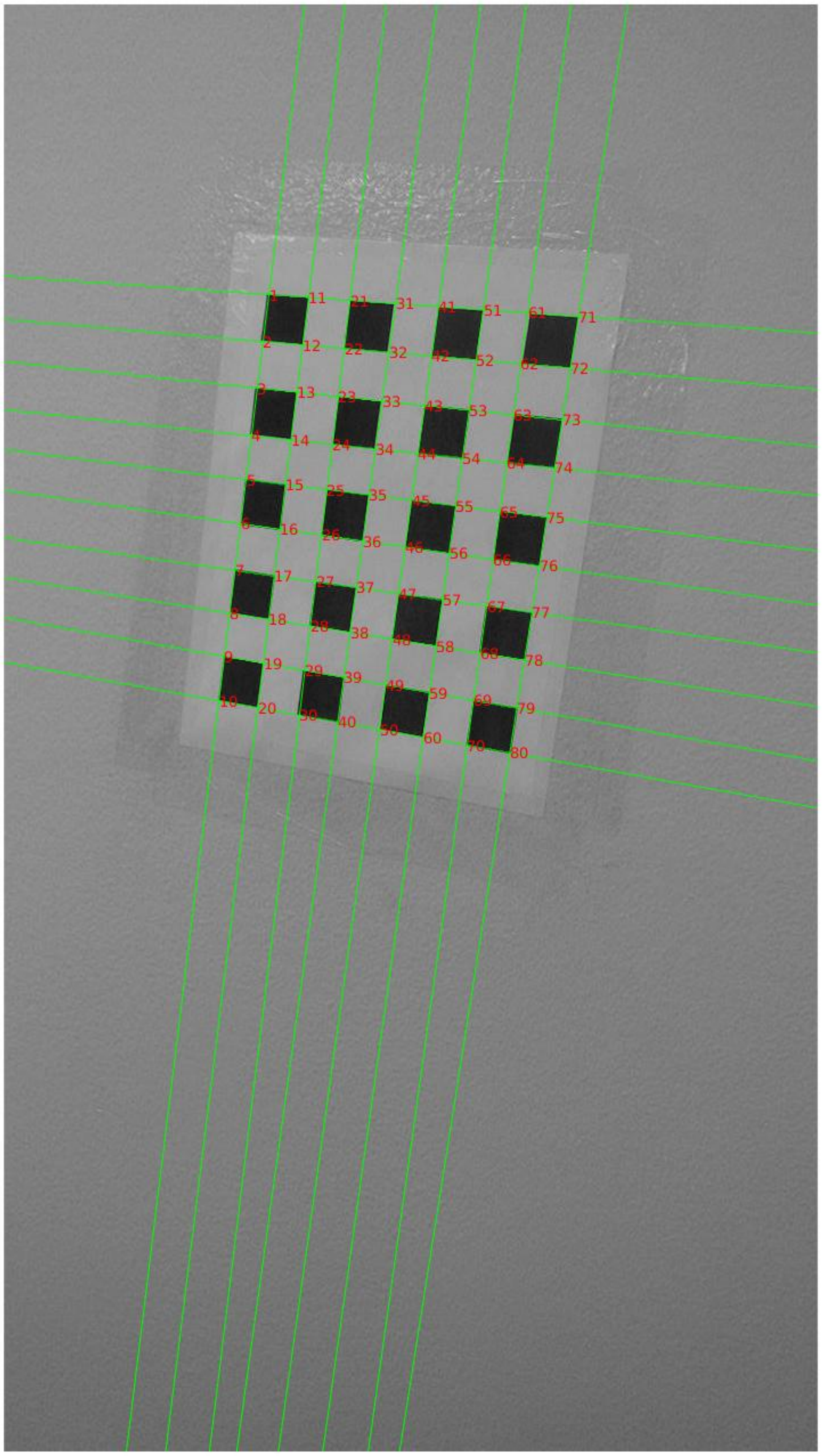

Figure 19: Corners for Pic 1 of Dataset

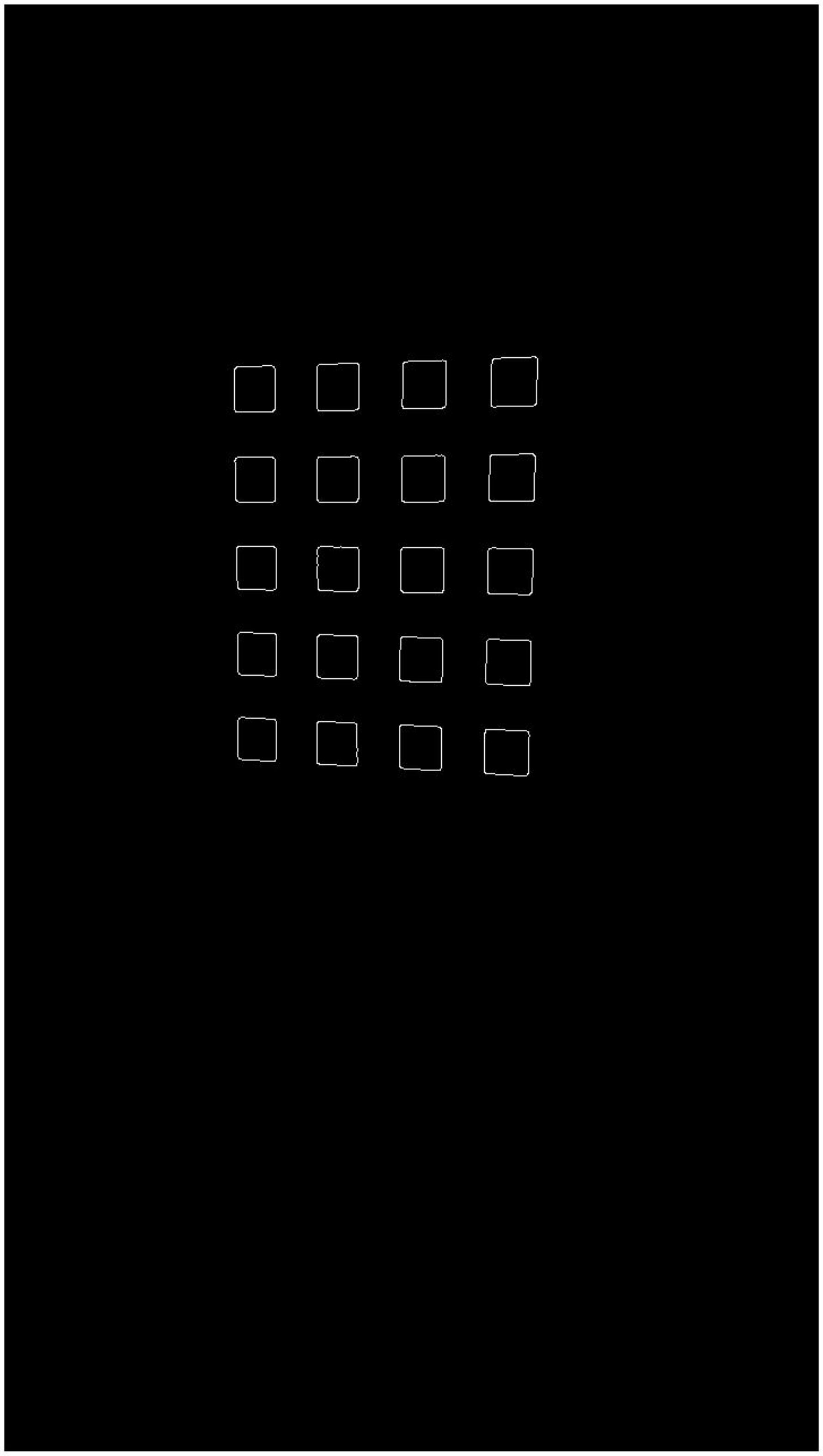

Figure 20: Canny Edge Detection Output for Pic 20 of Dataset

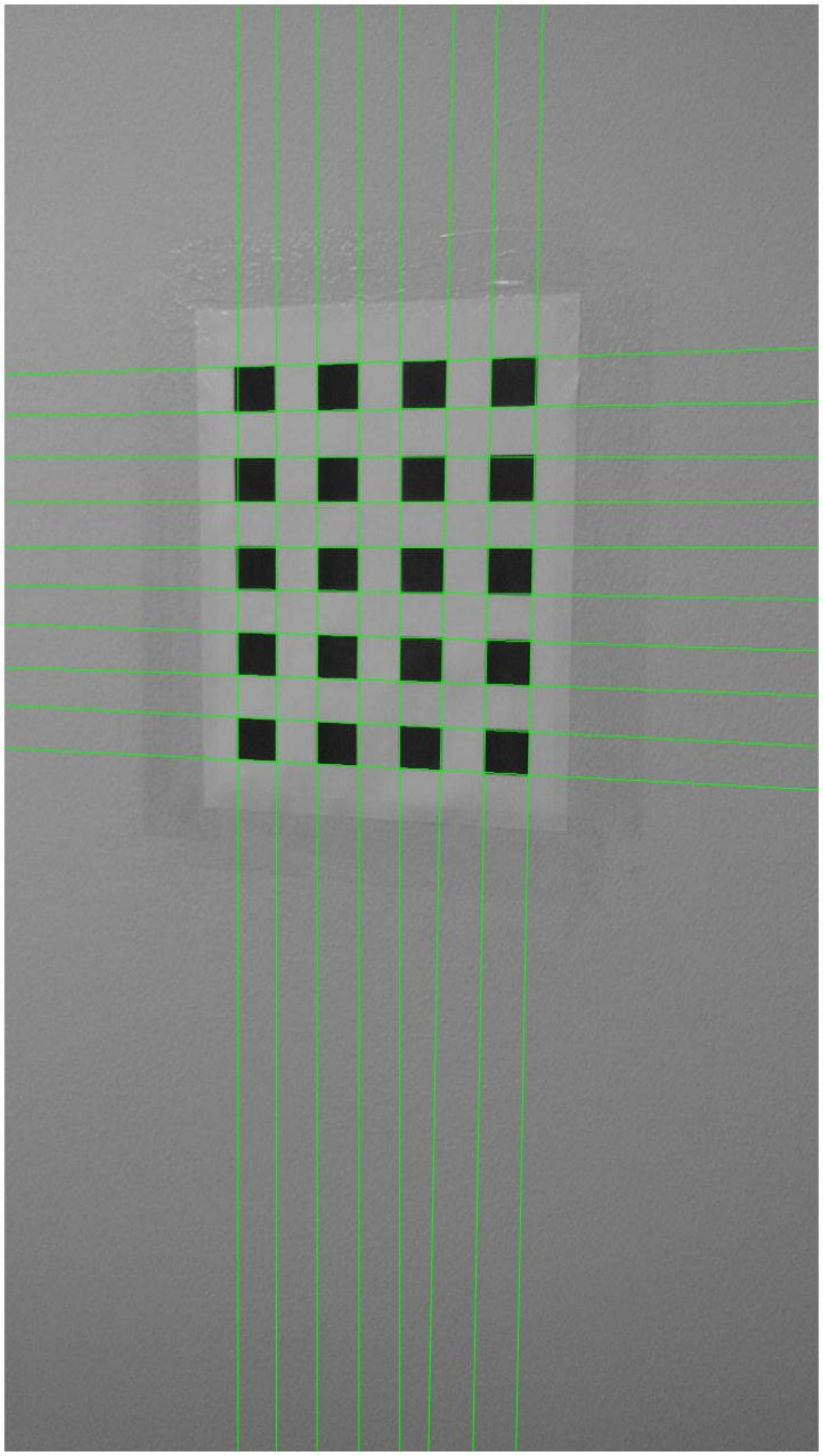

Figure 21: Hough Lines for Pic 20 of Dataset

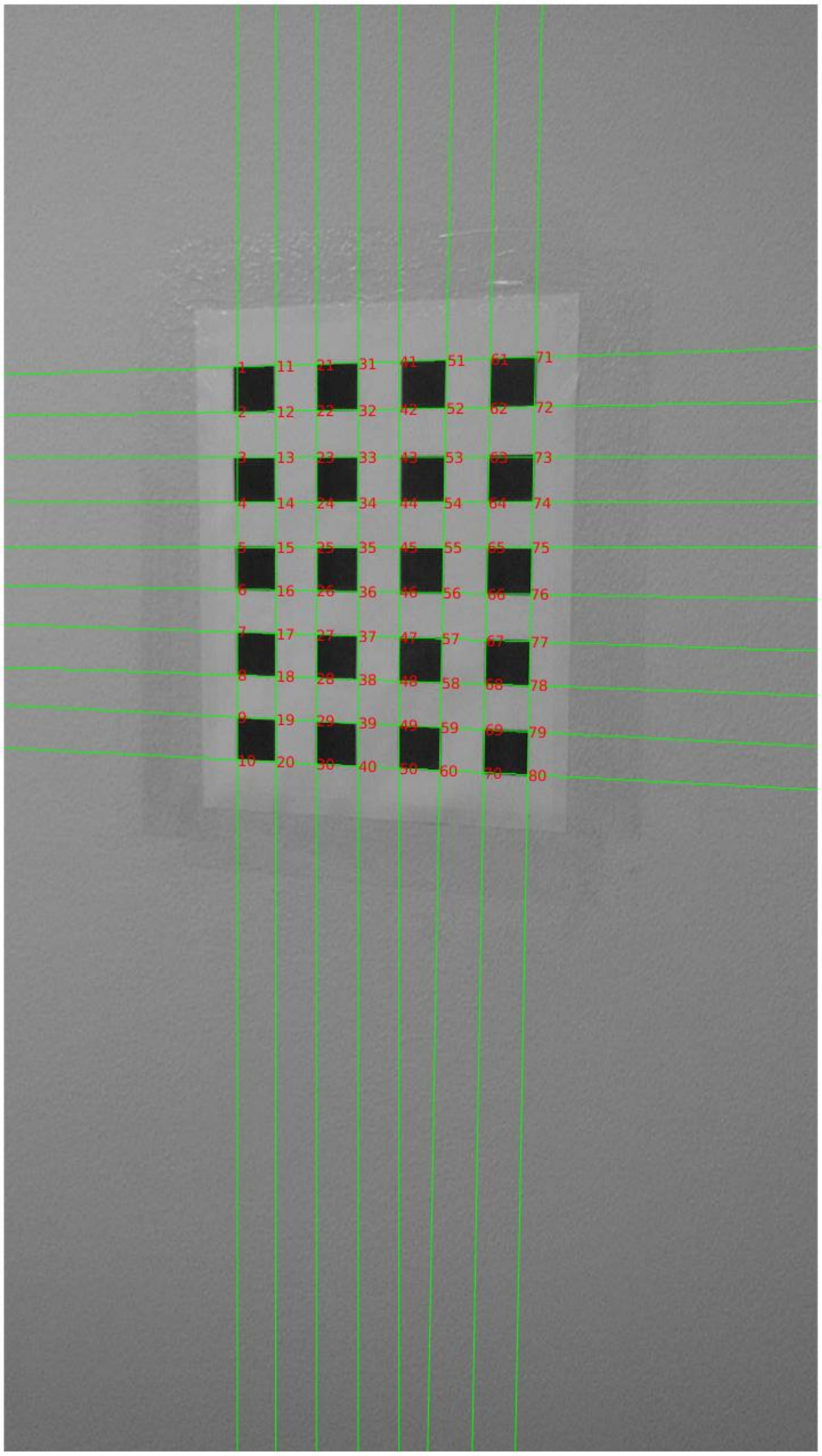

Figure 22: Corners for Pic 20 of Dataset

Images for extrinsic parameters

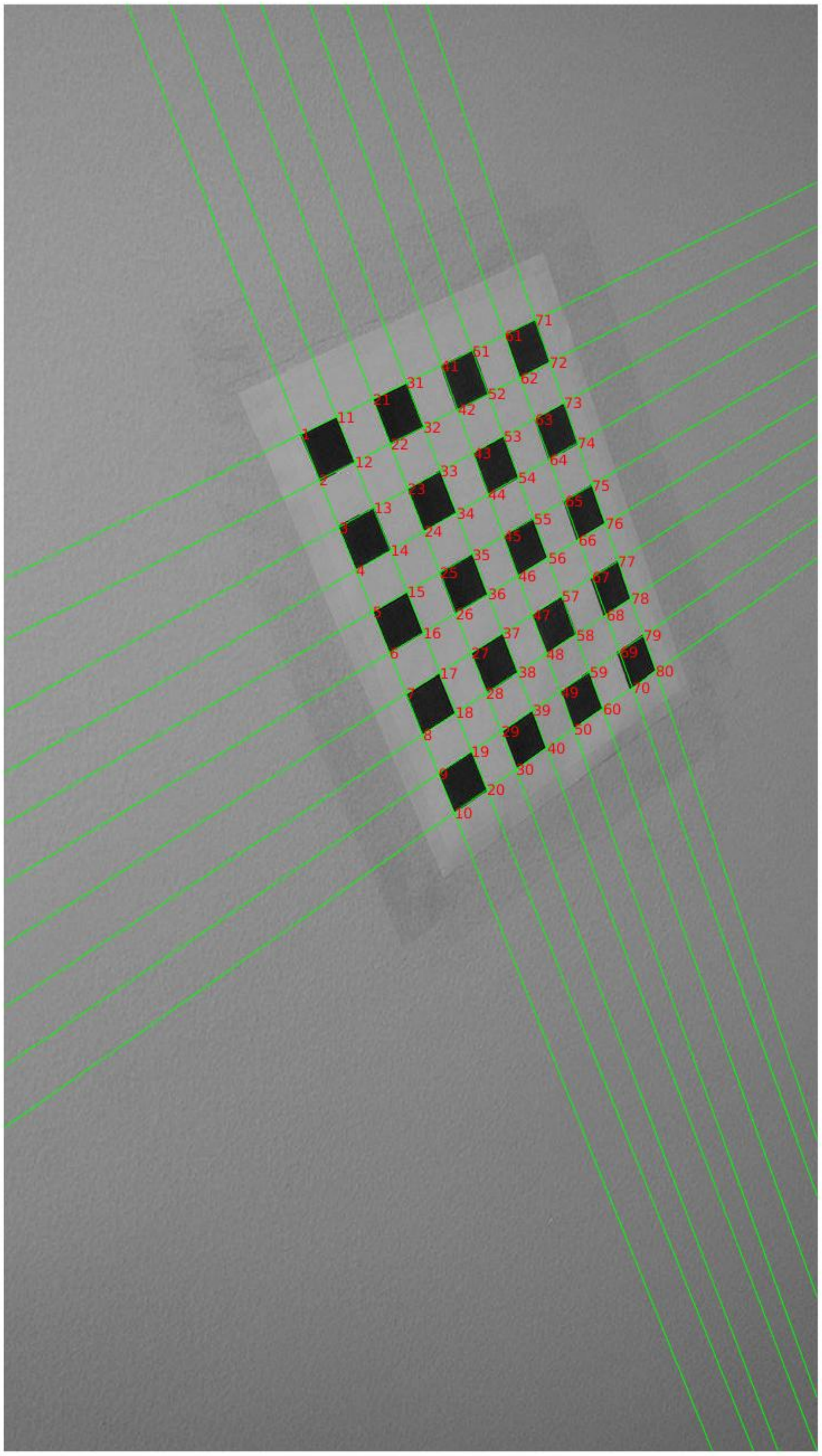

Figure 23: Extrinsic Parameters for Pic 2 of dataset

$$[R|t] = \begin{bmatrix} 0.7362 & 0.3656 & -0.6280 & -56.654 \\ -0.4685 & 0.8974 & -0.0485 & -143.325 \\ 0.555 & 0.297 & 0.8087 & 911.3086 \end{bmatrix}$$

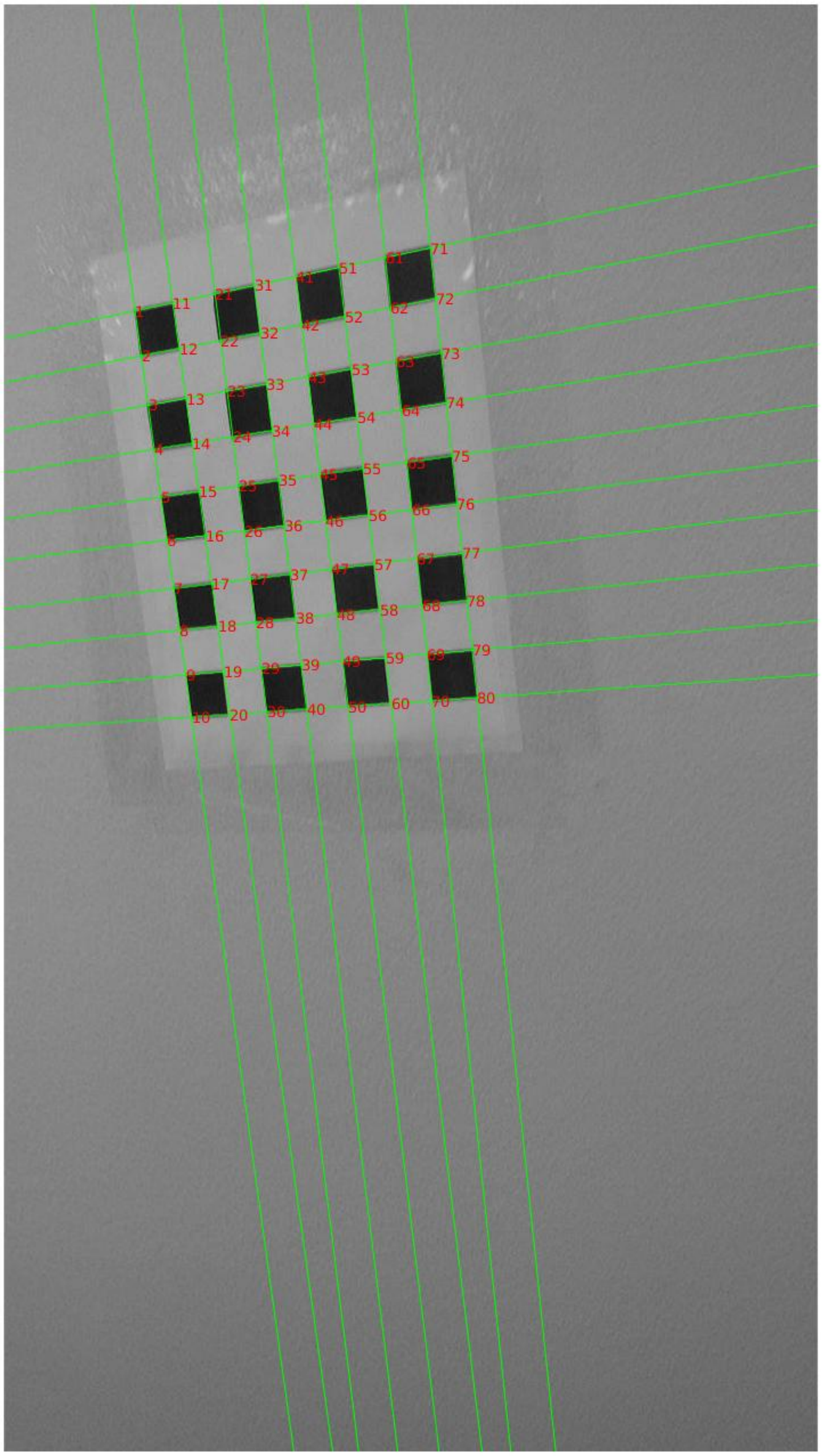

Figure 24: Extrinsic Parameters for Pic 3 of dataset

$$[R|t] = \begin{bmatrix} 0.9156 & 0.1145 & -0.5065 & -150.69 \\ -0.0604 & 0.9887 & -0.1828 & -217.90 \\ -0.5157 & 0.1547 & 0.9091 & 954.1858 \end{bmatrix}$$

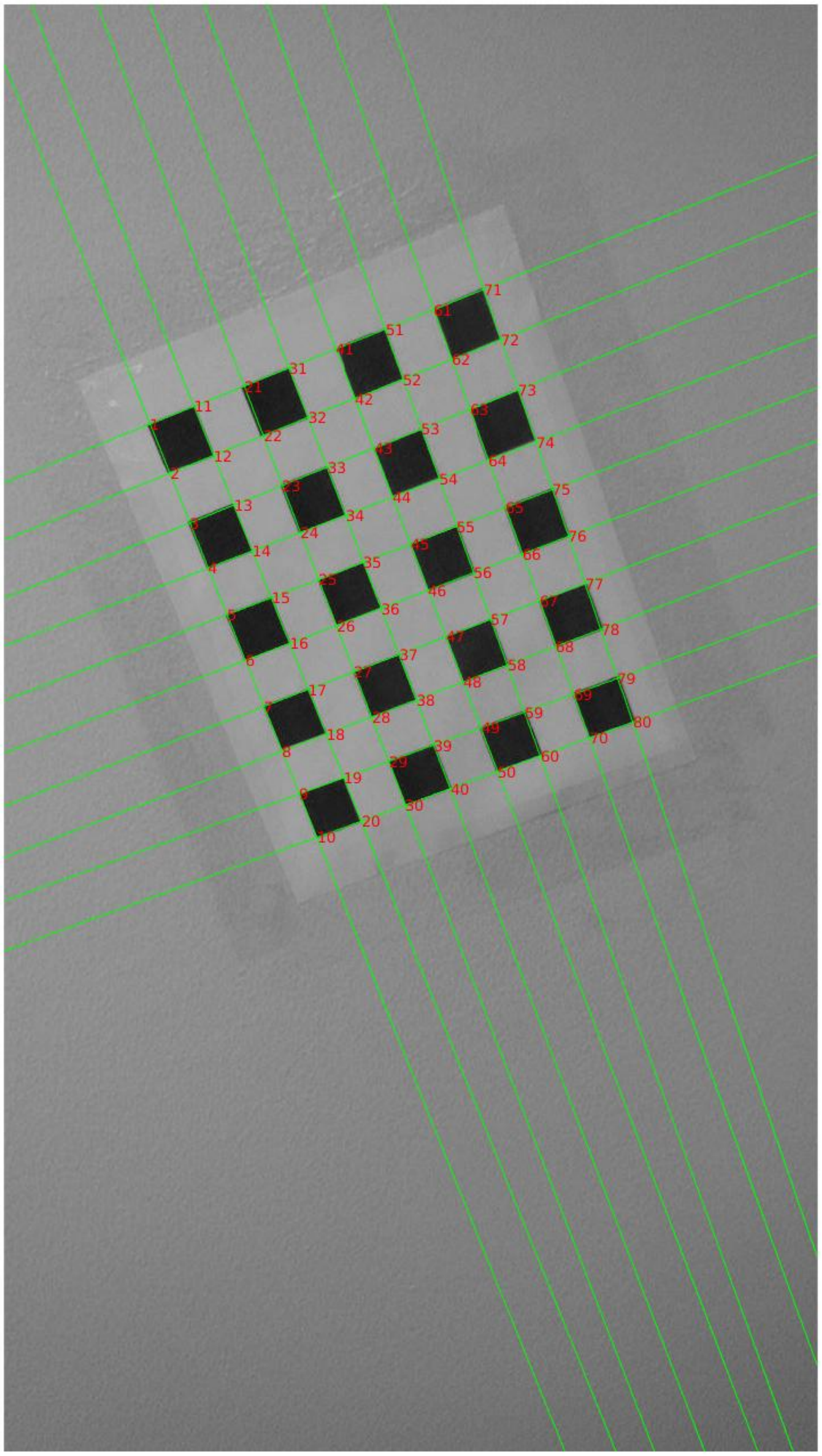

Figure 25: Extrinsic Parameters for Pic 7 of dataset

$$[R|t] = \begin{bmatrix} 0.9649 & 0.3711 & 0.0854 & -133.34 \\ -0.3593 & 0.9548 & -0.2389 & -145.31 \\ -0.1261 & -0.2202 & 0.9845 & 887.25 \end{bmatrix}$$

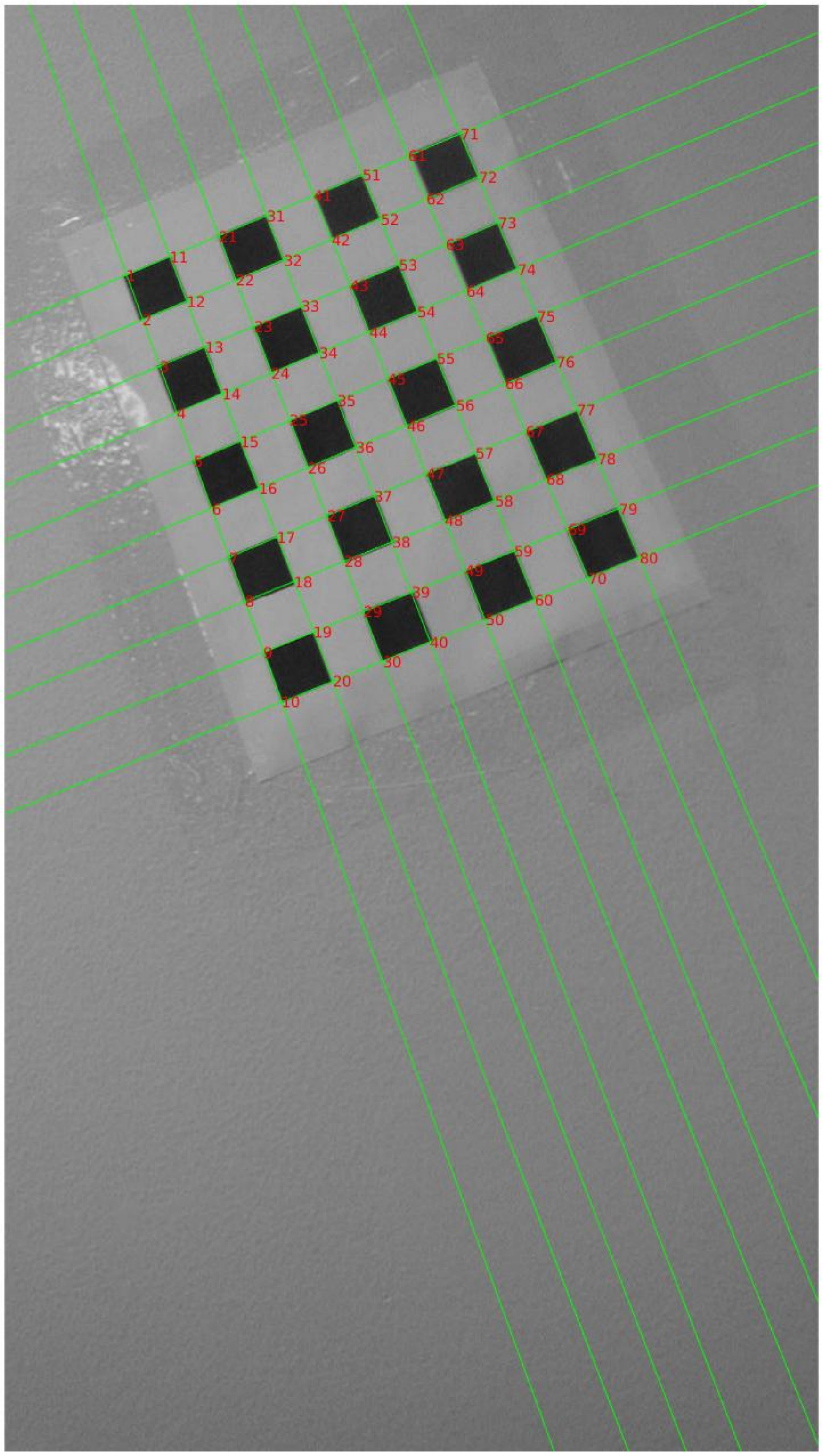
Figure 26: Extrinsic Parameters for Pic 10 of dataset

$$[R|t] = \begin{bmatrix} 0.9614 & 0.369 & 0.1388 & -144.5634 \\ -0.3823 & 0.952 & 0.2155 & -218.36 \\ -0.0964 & -0.2375 & 0.9838 & 881.4249 \end{bmatrix}$$

Output Images for Re-projection and also showing benefit of LM

We select Pic 4 of the dataset as the 'Fixed image'
Green markers are the ground truth and red markers are the projected ones

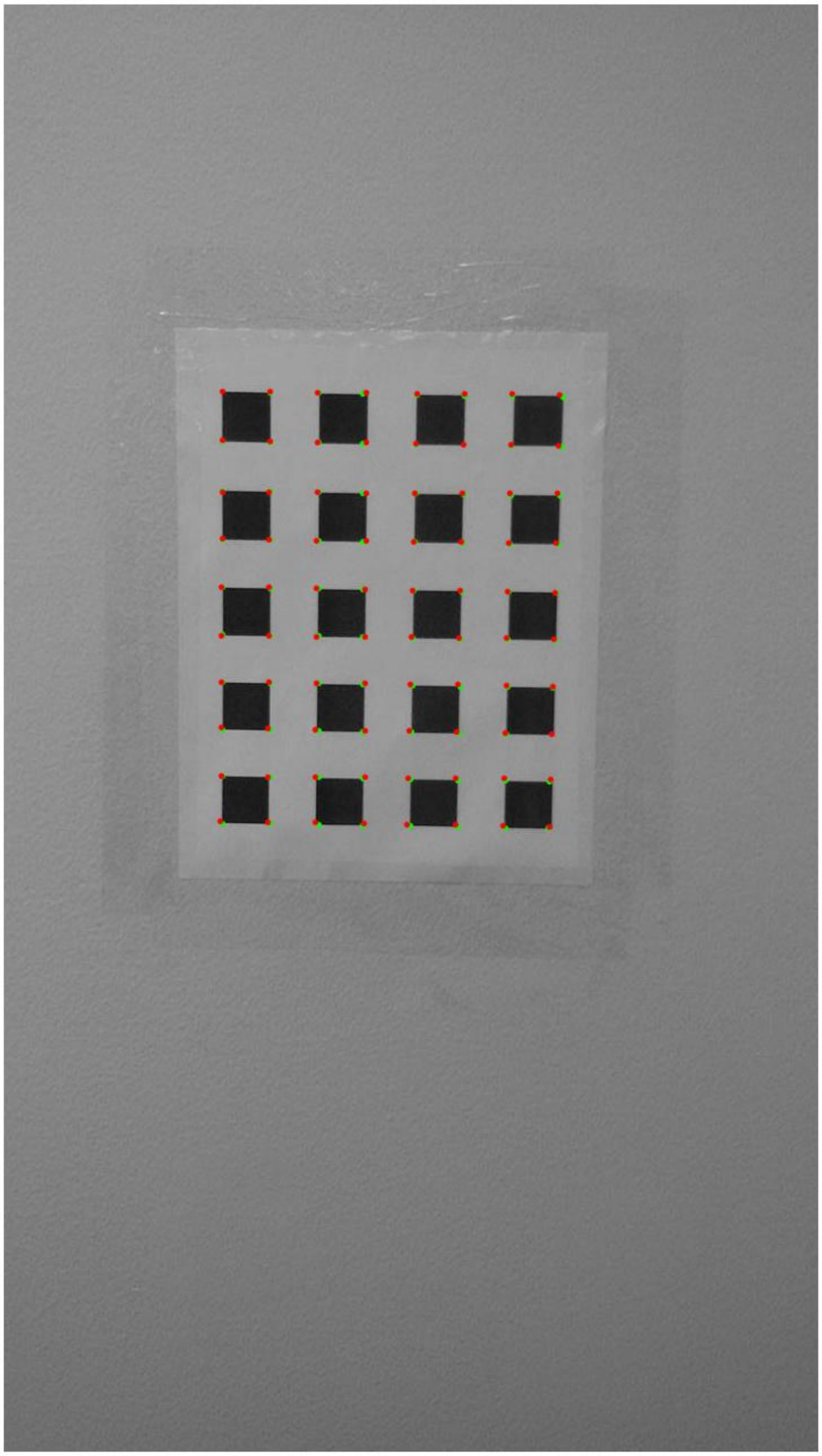

Figure 27: Re-projecting Pic 5 to 4 before LM (Error Mean = 1.1188, Error Variance = 0.6939)

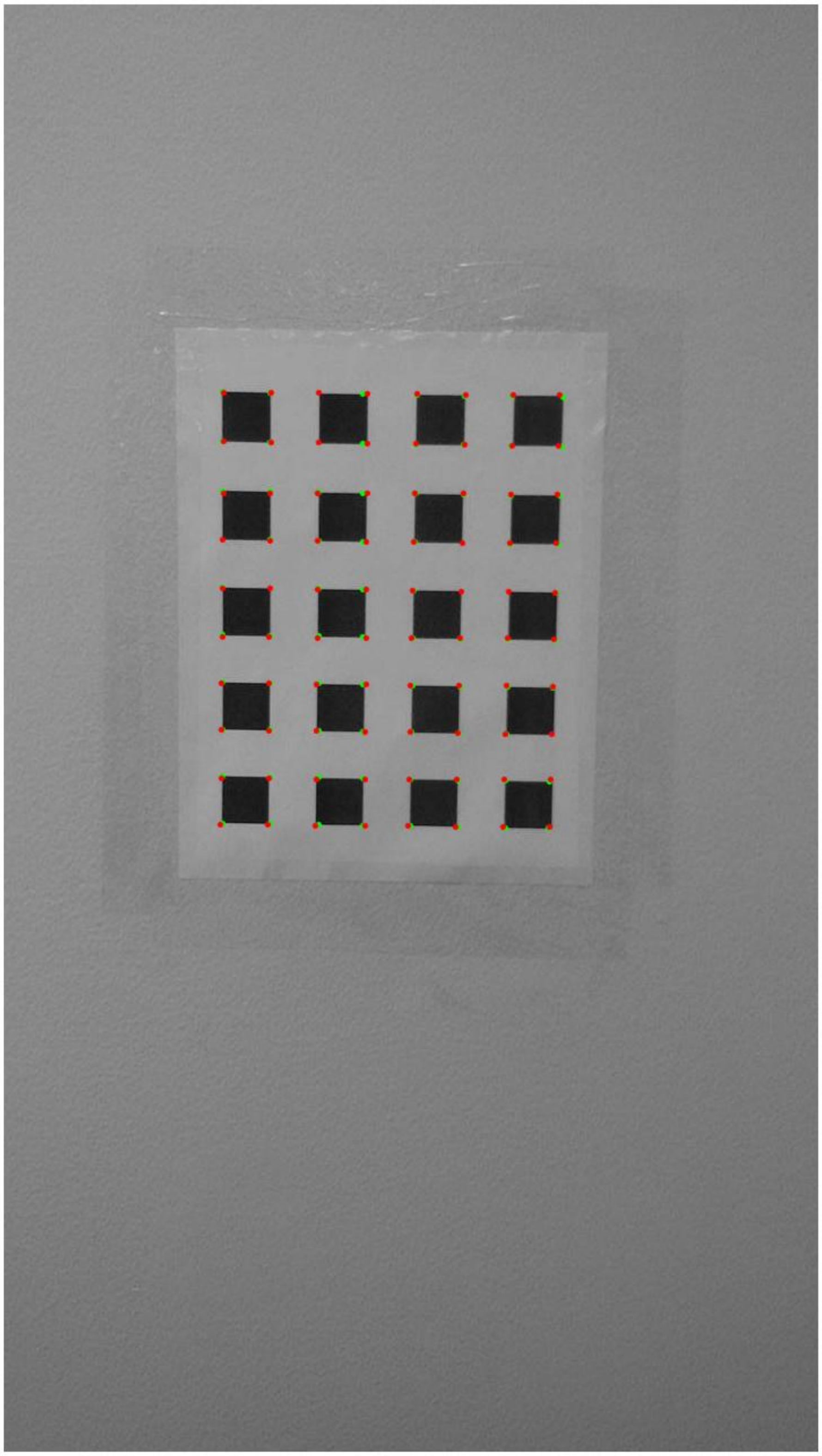

Figure 28: Re-projecting Pic 5 to 4 after LM(Error Mean = 0.9068, Error Variance = 0.5328)

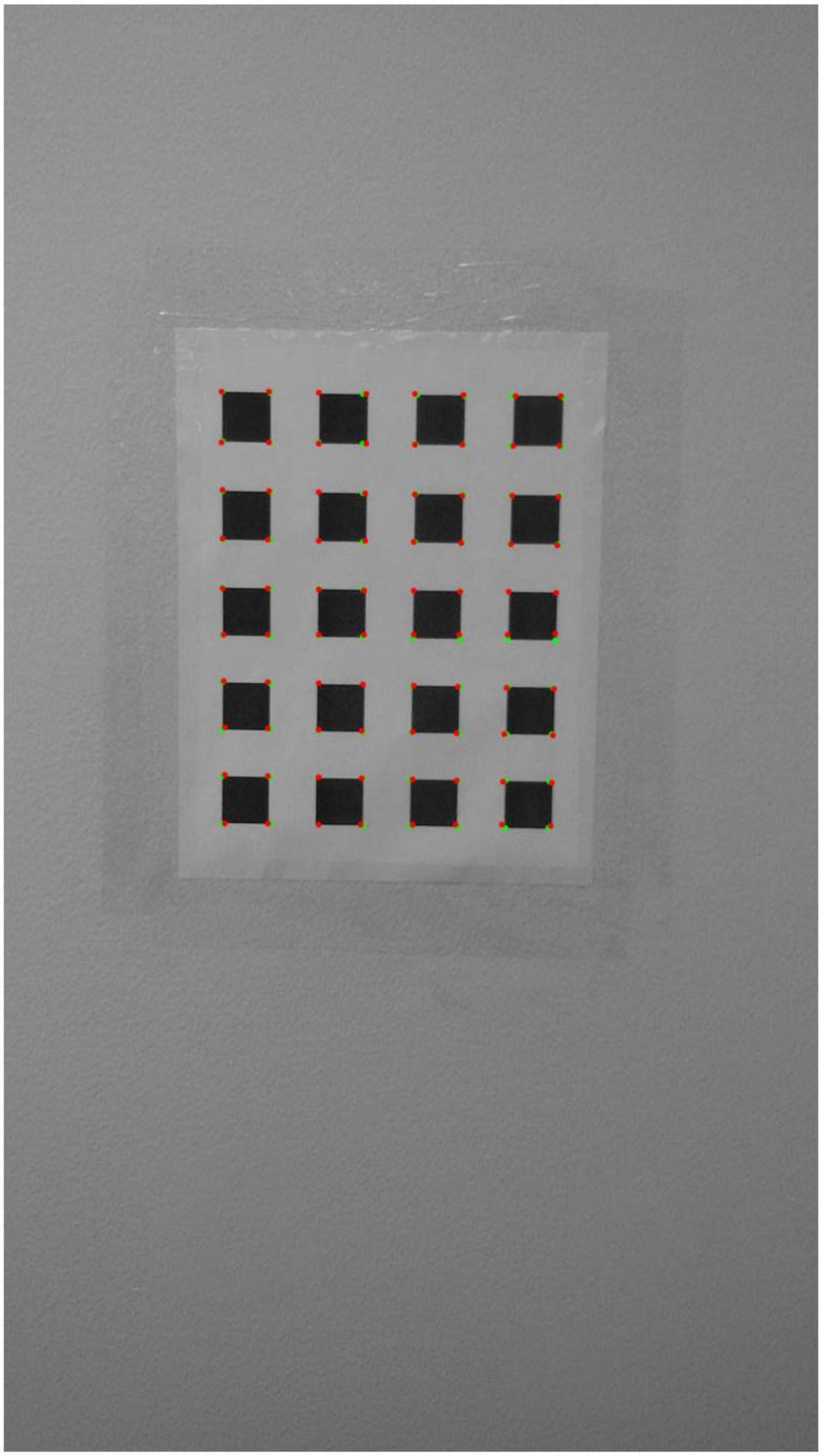

Figure 29: Re-projecting Pic 6 to 4 before LM (Error Mean = 1.2367, Error Variance = 0.9524)

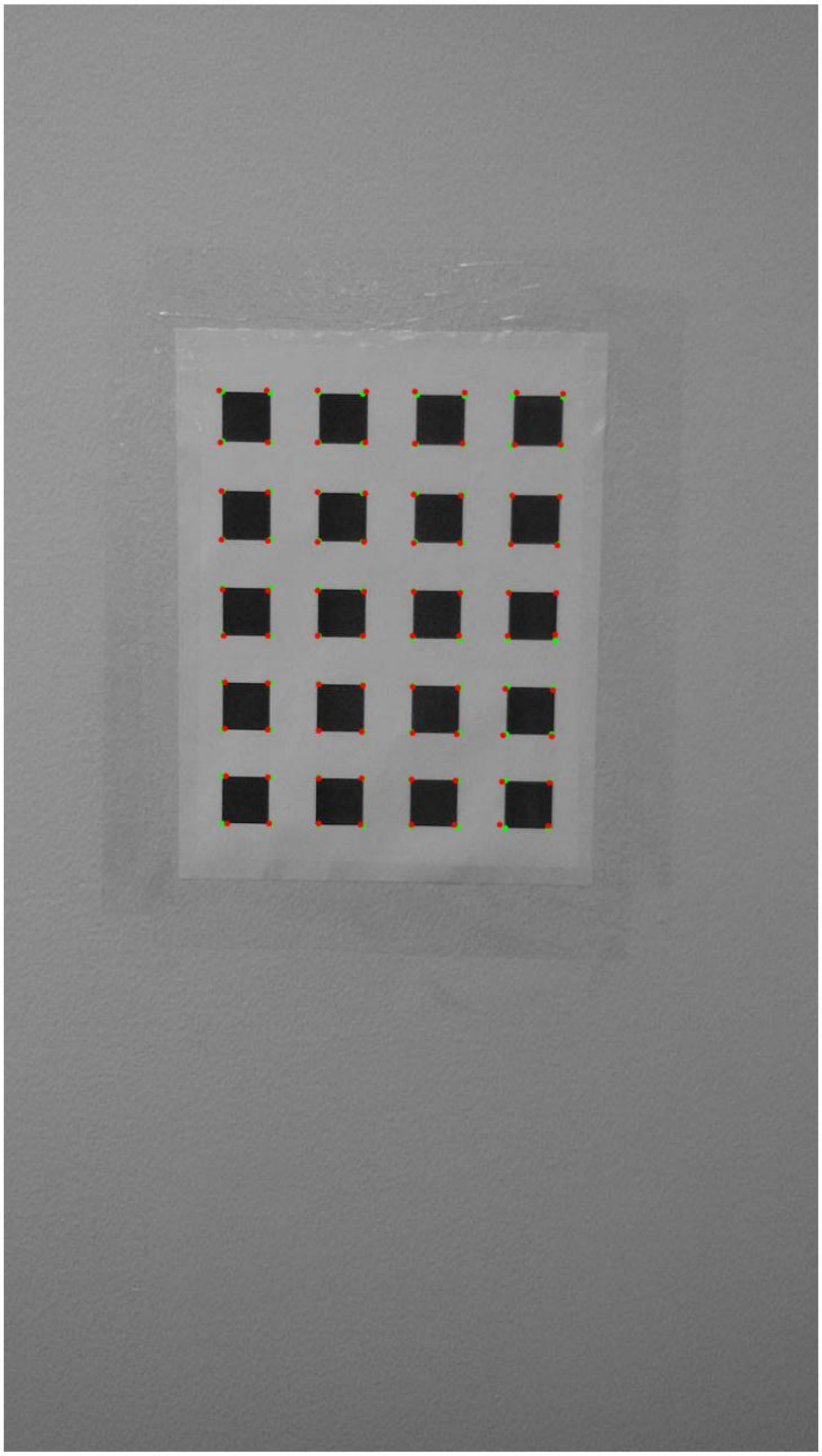

Figure 30: Re-projecting Pic 6 to 4 after LM (Error Mean = 0.9821, Error Variance = 0.8287)

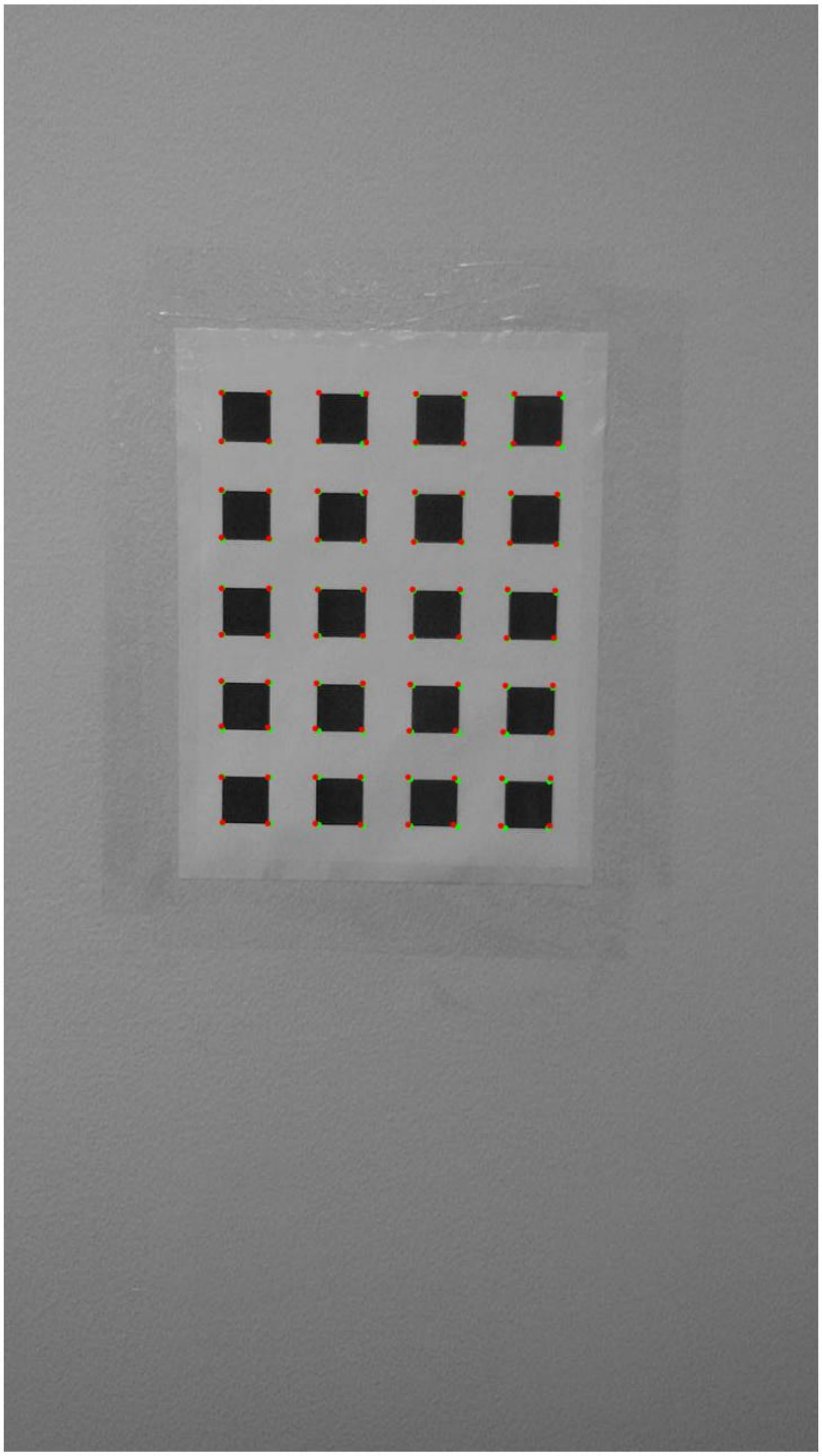

Figure 31: Re-projecting Pic 9 to 4 before LM (Error Mean = 1.3117, Error Variance = 0.8542)

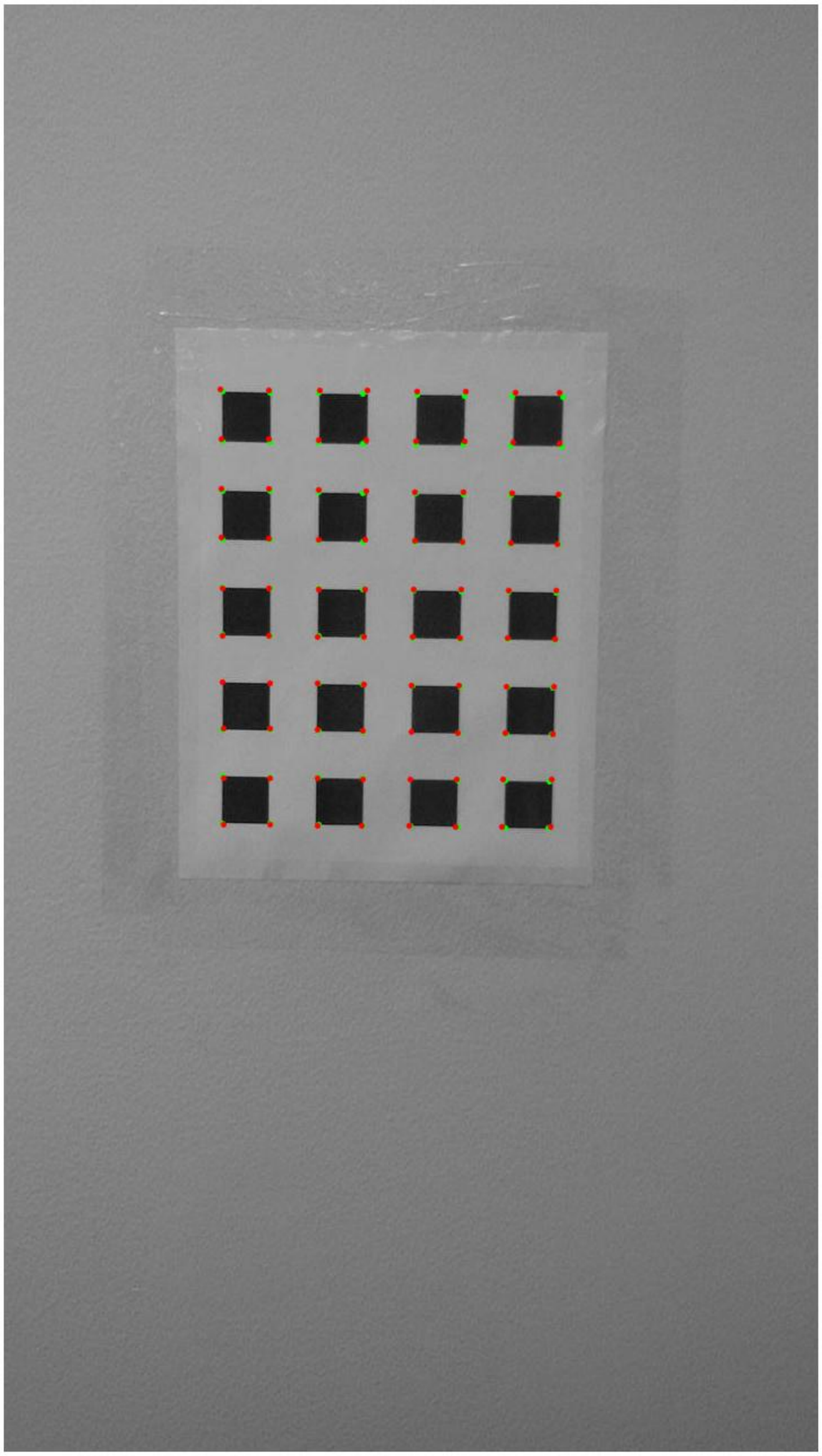

Figure 32: Re-projecting Pic 9 to 4 after LM (Error Mean = 1.1053, Error Variance = 0.6374)

Code

The script is in MATLAB 2016a and is self-explanatory

Script for Finding Corners

```
function [corner] = findCorners(filename)
gr_truth=imread(filename);
gr_truth_gray = rgb2gray(gr_truth);
gr_truth_edge = edge(gr_truth_gray,'canny',0.7);%for provided dataset 0.8
figure
imshow(gr_truth_edge)

[H, T, R] = hough(gr_truth_edge,'RhoResolution',0.5); %for provided dataset 0.5

P = houghpeaks(H,18,'Threshold',15); %for our and provided dataset 18 and 15
%There will be 18 lines
lines = houghlines(gr_truth_edge,T,R,P,'FillGap',150,'MinLength',70);
%for our and provided dataset 150 and 70
line_param = zeros(length(lines),2); %slope, y-intersect,
%Initializing the horizontal and vertical
hor = []; ver = [];


for k = 1:length(lines)
 xypt = [lines(k).point1; lines(k).point2];
 %find the equation of the line y = mx + b
 line_param(k,1) = (xypt(1,2)-xypt(2,2))/(xypt(1,1)-xypt(2,1));
 % plot_line(lines,k,size(gr_truth_edge));
 if(abs(line_param(k,1))>1)
 ver = [ver k];
 else
 hor = [hor k];
 end
 if(abs(line_param(k,1)) == inf)
 line_param(k,2) = inf;
 else
 line_param(k,2) = xypt(1,2) - line_param(k,1)*xypt(1,1);
 end
end
%Initializing the list for the corners
corner = [];
for i = 1:length(lines)
 n_c{i} = [];
end

%This is used to get rid of the extra lines
lines_hor = lines(hor);
ehor = zeros(1,length(hor));
for i= 1:length(lines_hor)
 for j = i+1:length(lines_hor)
 [pt]= intsect(lines_hor(i), lines_hor(j));
 if(pt(1)>1 && pt(1)<size(gr_truth,2) && pt(2)>1 && pt(2)<size(gr_truth,1))
 ehor(i) =ehor(i)+ 1;
 ehor(j) = ehor(j)+1;
 end
 end
end
lines_ver = lines(ver);
ever = zeros(1,length(ver));
for i= 1:length(lines_ver)
 for j = i+1:length(lines_ver)
 [pt]= intsect(lines_ver(i), lines_ver(j));
 if(pt(1)>1 && pt(1)<size(gr_truth,2) && pt(2)>1 && pt(2)<size(gr_truth,1))
 ever(i) = ever(i) +1;
 ever(j) = ever(j) +1;
 end
 end
end
%Sorting the line according to indices
[ever ind1] = sort(ever,'ascend');
[ever ind2] = sort(ehor,'ascend');
lines = lines([hor(ind2(1:10)) ver(ind1(1:8))]);

%plot the lines
figure
imshow(gr_truth_gray)
for k = 1:length(lines)
 ptxy = [lines(k).point1; lines(k).point2];
 %find the equation of the line y = mx + b
 %find slope m
 line_param(k,1) = (ptxy(1,2)-ptxy(2,2))/(ptxy(1,1)-ptxy(2,1));
 if(abs(line_param(k,1)) == inf)
 line_param(k,2) = inf;
 hold on
 y = 1:size(gr_truth,1);
 x = ptxy(1,1)*ones(1,length(y));
 plot(x,y,'Color','green')
 else
 line_param(k,2) = ptxy(1,2) - line_param(k,1)*ptxy(1,1);
 f = @(x) line_param(k,1)*x + line_param(k,2);
 x = 1:size(gr_truth,2);
 y = uint64(f(x));
 hold on
 plot(x,y,'Color','green');
 end
end

%*****************************************************************
%find the corners
for i= 1:length(lines)
 for j = i+1:length(lines)
 [pt]= intsect(lines(i), lines(j));
 if(pt(1)>1 && pt(1)<size(gr_truth,2) && pt(2)>1 && pt(2)<size(gr_truth,1))
 corner = [corner; pt ];
% hold on
% plot(pt(1),pt(2),'r*')
 n_c{i} = [n_c{i} size(corner,1)];
 n_c{j} = [n_c{j} size(corner,1)];
 end
 end
end

%label the corners same way
hor = [];
ver = [];
for i = 1:length(lines)
 if(length(n_c{i}) ==8)
 hor = [hor i];
 else
 ver = [ver i];
 end
end
xs = zeros(length(ver),1);
for i = 1:length(ver)
 %sort corners according to smallest x
 ind = n_c{ver(i)}; %these are corners that are on that line
 xs(i) = min(corner(ind,1)); %this is the smallest y for that vertical line
end
[d ind] = sort(xs,'ascend'); %sort vertical lines according to the smalles x
ver = ver(ind); %vertical lines are sorted
labels = zeros(80,1);
cnt = 0;
ys = zeros(10,1); %for each vertical line sort
for i = 1:length(ver)
 ind = n_c{ver(i)}; %these are corners that are on that line
 ys = corner(ind,2);
 [d sind] = sort(ys,'ascend');
 for j = 1:length(sind) %1 to 10
 cnt =cnt + 1;
 labels(cnt) = ind(sind(j));
 end
end
corner = corner(labels,:);

% This script is for labelling the corners
for i = 1:length(labels)
 hold on
 text(corner(i,1),corner(i,2),int2str(i),'Color','r');
end
end
```

Script for Finding Intersection of Lines

```
function [point]=intsect(l1, l2)
% This function is used to find the intersection of 2 lines
```

```
pt1 = [l1.point1 1];
pt2 = [l1.point2 1];
lA = cross(pt1,pt2); % Getting the first line
pt1 = [l2.point1 1];
pt2 = [l2.point2 1];
lB = cross(pt1,pt2); % getting the second line
point = cross(lA,lB); % getting the intersectiion of the two lines
point = double([point(1)/point(3) point(2)/point(3)]); % Converting to
% Homogeneous coordinates
end
```

Script for finding the least squares solution

```
function [A] = findA(xW,yW,xIM,yIM)
A = [];
for i = 1:length(xW)
 B = [xW(i) yW(i) 1 0 0 0 -xW(i)*xIM(i) -yW(i)*xIM(i) -xIM(i);
 0 0 0 xW(i) yW(i) 1 -xW(i)*yIM(i) -yW(i)*yIM(i) -yIM(i)];
 A = [A; B];
end
end
```

Script for defining objective function for LM algorithm

```
function err = dgeom(p,xW,xIM,rad_dist,nimg)
% p is the set of parameters for LM algorithm
ax = p(1);
s = p(2);
x0 = p(3);
ay = p(4);
y0 = p(5);
K = [ax s x0; 0 ay y0; 0 0 1]; % The intrinsic calibration matrix
if(rad_dist == 1)
 k1 = p(6);
 k2 = p(7); % These are the parameters of radial distortion
 K1 = [ax 0 x0; 0 ay y0; 0 0 1];
 cnt = 7;
else
 cnt = 5;
end
xproj = zeros(1,nimg*160);
n1=1;
for k = 1:nimg
 % Converting to the R,t using Rodriguez formula
 w = p(cnt+1:cnt+3);
 t = p(cnt+4:cnt+6)';
 cnt = cnt + 6;
 wx = [0 -w(3) w(2); w(3) 0 -w(1); -w(2) w(1) 0];
 phi = norm(w);
 R = eye(3)+sin(phi)/phi*wx + (1-cos(phi))/phi*wx^2;
 n2=1;
 for i = 1:80
 % Projection for all the corner points onto the fixed image.
 x = K*[R t]*[xW(n2:n2+1) 0 1]';
 xproj(n1:n1+1) = [x(1)/x(3) x(2)/x(3)];
 if(rad_dist == 1)
xp = [xproj(n1:n1+1) 1];
xw = inv(K1)*xp';
r2 = xw(1)^2 + xw(2)^2;
xp1 = xw(1) + xw(1)*(k1*r2+k2*r2^2);
xp2 = xw(2) + xw(2)*(k1*r2+k2*r2^2);
x = K1*[xp1 xp2 1]';
xproj(n1:n1+1) = [x(1)/x(3) x(2)/x(3)];
 end
 n1 = n1+2;
 n2 = n2+2;
 end
end
err = xIM - xproj;
end
```

Main Script

```
close all; warning off;
%Now we have to initialize the world co-ordinates measured with ruler
xW=zeros(80,2)
for j=1:8
    for i=1:10
        xW((j-1)*10+i,:)=[(j-1)*25 (i-1)*25];
    end
end

nimg=20; % The number of images used
rad_dist=1; % This is the indicator of whether radial distortion is used.

HAll=[];V=[];xIM=[];
for k =1:20
 filename = strcat('Dataset2/Pic_',int2str(k),'.jpg');
 %the coordinates of the corners in the image
 [imcoord] = findCorners(filename);
 xIM{k} = imcoord;
 %solve Ah = 0, and find A for the homography
 A = findA(xW(:,1),xW(:,2),double(imcoord(:,1)),double(imcoord(:,2)));
 [U,D,T] = svd(A);
 h = T(:,9);
 H = [h(1:3)'; h(4:6)'; h(7:9)'];
 HAll{k} = H;
 %V matrix for calculating the intrinsic parameters
 i=1;j=2;
 v12 = [H(1,i)*H(1,j), H(1,i)*H(2,j)+H(2,i)*H(1,j), H(2,i)*H(2,j), H(3,i)*H(1,j)+H(1,i)*H(3,j) ,H(3,i)
 i=1;j=1;
 v11 = [H(1,i)*H(1,j), H(1,i)*H(2,j)+H(2,i)*H(1,j), H(2,i)*H(2,j), H(3,i)*H(1,j)+H(1,i)*H(3,j) ,H(3,i)
 i=2;j=2;
 v22 = [H(1,i)*H(1,j), H(1,i)*H(2,j)+H(2,i)*H(1,j), H(2,i)*H(2,j), H(3,i)*H(1,j)+H(1,i)*H(3,j) ,H(3,i)
 V = [V;v12;(v11-v22)];
end

[U,D,T] = svd(V);
b = T(:,6); %B11 B12 B22 B13 B23 B33
%intrinsic parameters
y0 = (b(2)*b(4)-b(1)*b(5))/(b(1)*b(3)-b(2)^2);
lambda = b(6)-(b(4)^2+y0*(b(2)*b(4)-b(1)*b(5)))/b(1);
ax = sqrt(lambda/b(1));
ay = sqrt(lambda*b(1)/(b(1)*b(3)-b(2)^2));
s = -b(2)*ax^2*ay/lambda;
x0 = s*y0/ay-b(4)*ax^2/lambda;
K = [ax s x0; 0 ay y0; 0 0 1];

p = zeros(1,5+6*nimg);
p(1:5) = [ax s x0 ay y0];
if(rad_dist)
 p = zeros(1,7+6*nimg);
 p(1:5) = [ax s x0 ay y0];
 p(6:7) = [0 0];
 cnt = 7;
else
 p = zeros(1,5+6*nimg);
 p(1:5) = [ax s x0 ay y0];
 cnt = 5;
end

ydata=[];
K_inv = inv(K);
R_b4LM = [];
R_LM = [];
t_b4LM = [];
t_LM = [];

%This is for intrinsic parameters R,t
%The R is also converted to Rodriguez formula
%for w and theta
for k = 1:nimg
 H = HAll{k};
 t = K_inv*H(:,3);
 mag = norm(K_inv*H(:,1));
 if(t(3)<0)
 mag = -mag;
 end
 r1 = K_inv*H(:,1)/mag;
 r2 = K_inv*H(:,2)/mag;
 r3 = cross(r1,r2);
 R = [r1 r2 r3];
 t = t/mag;
 [U,D,V] = svd(R);
 R = U*V';
 R_b4LM{k}=R;
 t_b4LM{k}=t;
 % Rodriguez formula used here
 phi = acos((trace(R)-1)/2);
 w = phi/(2*sin(phi))*([R(3,2)-R(2,3) R(1,3)-R(3,1) R(2,1)-R(1,2)])';
 p(cnt+1:cnt+3) = w;
 p(cnt+4:cnt+6) = t;
 cnt = cnt + 6;
 y=xIM{k};
 y=y';
 ydata=[ydata y(:)'];
end
x = xW';
xdata = x(:)';

% LM algorithm is carried out for refinement
options = optimoptions('lsqcurvefit','Algorithm','levenberg-marquardt');
p1 = lsqnonlin(@dgeom,p,[],[],options,xdata,ydata,rad_dist,nimg);

% Finding the intrinsic calibration matrix
ax = p1(1);
s = p1(2);
x0 = p1(3);
ay = p1(4);
y0 = p1(5);
K1 = [ax s x0; 0 ay y0; 0 0 1];
if(rad_dist)
 k1 = p1(6);
 k2 = p1(7); % Finding the radial distortion parameters
 cnt = 7;
else
 cnt = 5;
end

%Converting back to R and t for extrinsic parameters after LM
for k = 1:nimg
 w = p1(cnt+1:cnt+3);
 t_LM{k} = p1(cnt+4:cnt+6)';
 cnt = cnt + 6;
 wx = [0 -w(3) w(2); w(3) 0 -w(1); -w(2) w(1) 0];
 phi = norm(w);
 R_LM{k} = eye(3)+sin(phi)/phi*wx + (1-cos(phi))/phi*wx^2;
end
```

Script for Reprojection

```
function [] = reproj(R,t,K,xW,k)
% R is the list of Calibrated rotation matrices
% t is the list of calibrated translation vectors
% K is the intrinsic camera calibration matrix
% xW is the list of corner world coordinates
% k is the index  of the projected image
```

```
%fixed image is image 4 for our dataset
filename = strcat('Dataset2/Pic_',int2str(4),'.jpg');
img = rgb2gray(imread(filename));
Pfix = K*[R{4}(:,1:2) t{4}]; %This is the Homography for the fixed image
xtrue = xW{4};
P = K*[R{k}(:,1:2) t{k}];%This is the Homography for the projected image
x0 = K(1,3);
y0 = K(2,3); % These are the co-ordinates of the principal point
xi = xW{k};
xi = [xi ones(size(xi,1),1)];
xyz = inv(P)*xi';
xest = (Pfix*xyz)';
figure
imshow(img)
%Now plotting the reprojections
for i = 1:80
 xest(i,:) = xest(i,:) / xest(i,3);
 hold on
 plot(uint64(xtrue(i,1)),uint64(xtrue(i,2)),'g.','MarkerSize',12);
 hold on
 plot(uint64(xest(i,1)),uint64(xest(i,2)),'r.','MarkerSize',12);
end
xest = xest(:,1:2);
hold off
mean(abs(xtrue(:)-xest(:))); %Plotting the mean of error
var(abs(xtrue(:)-xest(:))); %Plotting the variance of error
end
```