COMPUTER VISION

ECE 661

HOMEWORK 9

Tanzeel U. Rehman
Email: *trehman@purdue.edu*

1. Introduction:

This homework performs runs the camera calibration routines on the set of images containing the checkboard pattern. Two datasets were used for this homework. One dataset was provided with this homework and other was created as the homework instructions. The implemented algorithm has following steps:

a) First detect the edges of all the boxes in a checkerboard pattern using the Canny edge detector. We then use the Hough transformation to extract the horizontal and vertical lines joining the edges. The intersection of the lines yields corners in an image.

b) Then we calculated the homography matrices between image coordinates and the world coordinates of the boxes on checkerboard pattern. These homographies were finally used with Zhang's algorithm [1)] to estimate the intrinsic and extrinsic parameters of the camera and images, respectively.

c) The intrinsic and extrinsic parameters were then refined using the Levenburg-Marquardt (LM) algorithm to compensate for the non-linearities.

1.1. Description of Methods:

A. Corners detection:

For this homework we used MATLAB as scripting language. Each image was converted to the grayscale and then the Canny edge detector function was applied to extract the boxes in checkerboard pattern. The edge detector threshold of 0.8 was used for two datasets. The edges were used to find the horizontal and vertical lines connecting them with the help of Hough transform function in the MATLAB. The "rho" parameter of the MATLAB function was set to the 0.55 and 0.65 for the given and extracted dataset. For a pair of horizontal and vertical line, we

found the intersection, thus yielding the corner belonging to that specific pair. We extracted all the corners for every line pair extracted using Hough transformation.

B. Camera calibration:

The image corners (x_i) measured in the above step together with their real-world coordinates (x_w) were used for estimating the homographies. The box size in the checkerboard pattern was measured in physical world with the help of ruler and real-world coordinates were extracted using the physical size with the top left corner of left most corner have coordinates of (0,0). The homography for each image of the form $x_i = Hx_w$ was solved using least squares solution. For estimating the intrinsic parameters (*K),* we used the absolute conic method. The camera image of absolute conic (ω) is independent of the extrinsic parameters i.e., it is solely dependent on the intrinsic parameters and can be written as Eq.1.

$$\omega = K^{-T}K^{-1} \tag{Eq.1}$$

Now, if we write the *H* as Eq. 2, then we can show the following.

$$H = [\vec{h}_1\ \vec{h}_2\ \vec{h}_3] \tag{Eq.2}$$

$$\vec{h}_1{}^T\,\omega\,\vec{h}_1 - \vec{h}_2{}^T\,\omega\,\vec{h}_2 = 0 \tag{Eq.3}$$

$$\vec{h}_1{}^T\,\omega\,\vec{h}_2 = 0 \tag{Eq.4}$$

The Eq.4 can be represented in terms of vectors $\vec{b}$ of unknowns as shown in Eq. 5 and using the same logic the Eq. 3 can be presented in terms of vectors $\vec{b}$ of unknowns as Eq. 6.

$$\vec{h}_1{}^T\,\omega\,\vec{h}_2 = (\vec{h}_1\ \vec{h}_2\ \vec{h}_3)\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}\begin{bmatrix} \vec{h}_1 \\ \vec{h}_2 \\ \vec{h}_3 \end{bmatrix} = \vec{v}_{12}{}^T\vec{b} = 0 \tag{Eq.5}$$

$$\vec{h}_1{}^T \omega \vec{h}_1 - \vec{h}_2{}^T \omega \vec{h}_2 = (\vec{v}_{11} - \vec{v}_{22})\vec{b} = 0 \quad \text{(Eq.6)}$$

Where $\vec{v}_{ij}$ can be expressed using Eq. 7. The Eqs. 5 and 6 can be combined as Eq.8.

$$v_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix} \quad \text{(Eq.7)}$$

$$\begin{bmatrix} \vec{v}_{12}{}^T \\ (\vec{v}_{11} - \vec{v}_{22}) \end{bmatrix} \vec{b} = V\vec{b} = 0 \quad \text{(Eq.8)}$$

For each i^{th} position of camera, we can get the V (2 × 6) matrix, which can be stacked vertically for n number of camera positions to give us the equation of the form $\begin{bmatrix} V_1 \\ V_2 \\ \dots \\ V_n \end{bmatrix} \vec{b} = 0$. These equations can finally be solved using system of linear least squares as discussed in the Lecture 11 of this course to get the vector $\vec{b}$. Once the vector $\vec{b}$ is computed, the K matrix can be formulated as Eq .9.

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(Eq.9)}$$

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

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

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

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

$$s = -\frac{w_{12}\alpha_x\alpha_y}{\lambda}$$

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

Now, the intrinsic parameters for all the images are same, however, the extrinsic parameters including rotation *R* and translation *t* were different. Therefore, these parameters we computed individually using the relationship in Eq. 10. The *R* and *t* can be given by Eqs. 11 and 12.

$$K^{-1}\left[\vec{h}_1\ \vec{h}_2\ \vec{h}_3\right] = \left[\vec{r}_1\ \vec{r}_2\ \vec{t}\right] \quad \text{(Eq.10)}$$

$$R = [\vec{r}_1\ \vec{r}_2\ \vec{r}_2] \quad \text{(Eq.11)}$$

$$\vec{r}_1 = \epsilon K^{-1}\vec{h}_1$$

$$\vec{r}_2 = \epsilon K^{-1}\vec{h}_2$$

$$\vec{r}_3 = \vec{r}_1 \times \vec{r}_2$$

Where,

$$\epsilon = scale = \frac{1}{\left\|K^{-1}\vec{h}_1\right\|}$$

$$\vec{t} = \epsilon K^{-1}\vec{h}_3 \quad \text{(Eq.12)}$$

The rows of *R* matrix were conditioned to be orthonormal by decomposing it using SVD ($R = UDV$) and resetting all the singular values to 1 by making $R = UV^T$.

C. Refining the Camera calibration:

The intrinsic and extrinsic parameters estimated above based on the linear least squares can further be refined using the non-linear LM optimization algorithm. The cost function needed for refining was defined as in Eq. 13.

$$d_{geom}^2 = \sum_i \sum_j \left\| x_{ij} - K\left[R_i | \vec{t}_i\right] \vec{x}_{M,j} \right\|^2 \quad \text{(Eq.13)}$$

Where, $\vec{x}_{M,j}$ is the j^{th} point on the calibration pattern, x_{ij} is the actual image point for $\vec{x}_{M,j}$in the i^{th} position of the camera, *K* is intrinsic parameters matrix and R_i and $\vec{t}_i$ are the extrinsic parameters for the i^{th} image. Since the actual *R* matrix has 9 elements and 3 DOFs, there it can't be minimized using LM algorithm as this optimization strictly requires the number of elements to be equal to DOFs. Therefore, the *R* matrix was reparametrized to obtain 3 parameter representation (ω) using the Rodrigues representation. Let the 9 elemental *R* be represented by Eq. 14, then ω can be obtained using Eq. 15. After refining the parameters using LM, the *R* can again be represented in elemental representation using Eq 16.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad \text{(Eq.14)}$$

$$\omega = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = \frac{\phi}{2sin\phi} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{12} - r_{12} \end{bmatrix}, \quad \text{(Eq.15)}$$

$$\phi = cos^{-1}\phi \frac{trace(R) - 1}{2}$$

$$\mathrm{R} = I + \frac{sin\phi}{\phi}[\vec{\omega}]_x + \frac{1 - cos\phi}{\phi^2}[\vec{\omega}]_x^2, \quad \text{(Eq.16)}$$

$$[\vec{\omega}]_x = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}$$

D. Incorporating the radial distortion:

The camera calibration done so far is for the pinhole camera. In case of short focal length camera, there appears to be radial distortion in the images induced by the camera lens, which leaves barrel

or pincushion effect in the images (straight lines appear to be the curves). This distortion can be corrected by using the Eq. 17.

$$\hat{x}_{rad} = \hat{x} + (\hat{x} - x_0)[k_1 r^2 + k_2 r^4] \quad \text{(Eq.6)}$$

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

Where, $(\hat{x}, \hat{y})$ are the coordinates of a point obtained through pinhole camera model, $(\hat{x}_{rad}, \hat{y}_{rad})$ is the coordinates of the same point after correcting the radial distortion, (x_0, y_0) are the coordinates of principal point, k_1 and k_2 are the radial distortion parameters and $r = (\hat{x} - x_0)^2 + (\hat{y} - y_0)^2$.

1.2. Dataset 1:

1.2.1. Intrinsic parameters for the given dataset:

(a) Linear least-squares (LLS):

$$K = \begin{bmatrix} 723.4997 & 1.4841 & 235.2641 \\ 0 & 722.1189 & 319.4149 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) LM algorithm refined without radial distortion:

$$K = \begin{bmatrix} 874.0656 & 1.5920 & 320.1941 \\ 0 & 874.1790 & 233.8322 \\ 0 & 0 & 1 \end{bmatrix}$$

(c) LM algorithm refined with radial distortion:

k1 = -0.17704

k2 = 0.59277

$$K = \begin{bmatrix} 878.0351 & 1.6382 & 318.6801 \\ 0 & 878.6688 & 232.2903 \\ 0 & 0 & 1 \end{bmatrix}$$

1.2.2. Extrinsic parameters for the given dataset:

(a) Image 9 using LLS and LM:

$$\left[R_9 | \vec{t}_9\right]_{LLS} = \begin{bmatrix} 0.8840 & -0.1378 & 0.4467 & 19.5082 \\ -0.1261 & 0.8498 & 0.5118 & -170.1235 \\ -0.4502 & -0.5088 & 0.7388 & 633.9512 \end{bmatrix}$$

$$[R_9|\vec{t}_9]_{LM} = \begin{bmatrix} 0.8886 & -0.0882 & 0.4844 & -52.3874 \\ -0.1761 & 0.8409 & 0.5548 & -88.8184 \\ -0.4598 & -0.5754 & 0.7314 & 709.3040 \end{bmatrix}$$

(b) Image 12 using LLS and LM:

$$[R_{12}|\vec{t}_{12}]_{LLS} = \begin{bmatrix} 0.9036 & 0.2036 & -0.3769 & -4.6823 \\ -0.0307 & 0.9083 & 0.4171 & -170.1357 \\ -0.4273 & -0.3653 & 0.8270 & 511.4145 \end{bmatrix}$$

$$[R_{12}|\vec{t}_{12}]_{LM} = \begin{bmatrix} 0.8869 & 0.2216 & -0.4653 & -68.6000 \\ -0.0140 & 0.8931 & 0.5014 & -115.1866 \\ -0.5151 & -0.4500 & 0.7923 & 652.2641 \end{bmatrix}$$

(c) Image 25 using LLS and LM:

$$[R_{25}|\vec{t}_{25}]_{LLS} = \begin{bmatrix} 0.9438 & -0.2967 & -0.1455 & 14.3185 \\ 0.2818 & 0.9526 & -0.1147 & -182.8481 \\ 0.1727 & 0.0673 & 0.9827 & 459.7933 \end{bmatrix}$$

$$[R_{25}|\vec{t}_{25}]_{LM} = \begin{bmatrix} 0.9691 & -0.3185 & -0.2102 & -41.8313 \\ 0.3087 & 0.9768 & -0.1202 & -134.2409 \\ 0.2242 & 0.0913 & 0.9876 & 579.5434 \end{bmatrix}$$

(d) Image 35 using LLS and LM:

$$[R_{35}|\vec{t}_{35}]_{LLS} = \begin{bmatrix} 0.9524 & -0.0598 & -0.2988 & -0.7612 \\ 0.0384 & 0.9963 & -0.0770 & -151.6005 \\ 0.3023 & 0.0618 & 0.9512 & 514.4299 \end{bmatrix}$$

$$[R_{35}|\vec{t}_{35}]_{LM} = \begin{bmatrix} 0.9461 & -0.0796 & -0.4522 & -62.8548 \\ -0.0647 & 0.9976 & -0.0728 & -92.6590 \\ -0.4546 & -0.0561 & 0.9464 & 632.2045 \end{bmatrix}$$

1.2.3. Extrinsic parameters with LM and distortion incorporated:

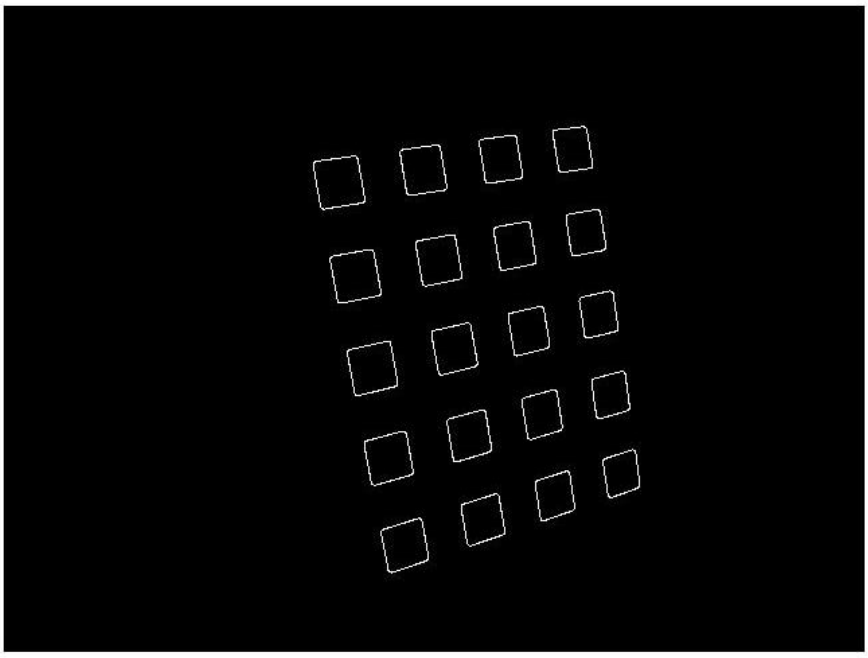
$$[R_{35}|\vec{t}_{35}]_{LM} = \begin{bmatrix} 0.9466 & -0.0793 & -0.4513 & -62.8548 \\ -0.0659 & 0.9978 & -0.0663 & -92.6590 \\ -0.4534 & -0.0496 & 0.9471 & 632.2045 \end{bmatrix}$$

1.2.4. Camera pose for fixed imaged with measured ground-truth :

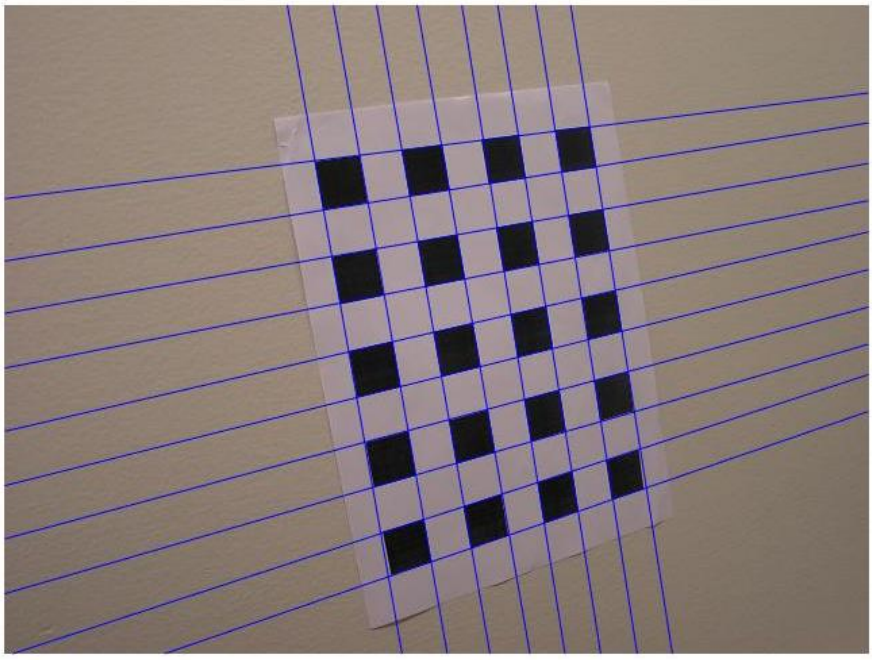
The $[R_{11}|\vec{t}_{11}]$for the fixed image 11 with LLS and LM are given below. It can be seen that R is approximately identity matrix with t representing the translation.

$$[R_{11}|\vec{t}_{11}]_{LLS} = \begin{bmatrix} 0.9994 & -0.0072 & 0.0342 & 19.5082 \\ 0.0088 & 0.9989 & -0.0469 & -170.1235 \\ -0.0338 & 0.0472 & 0.9983 & 633.9512 \end{bmatrix}$$

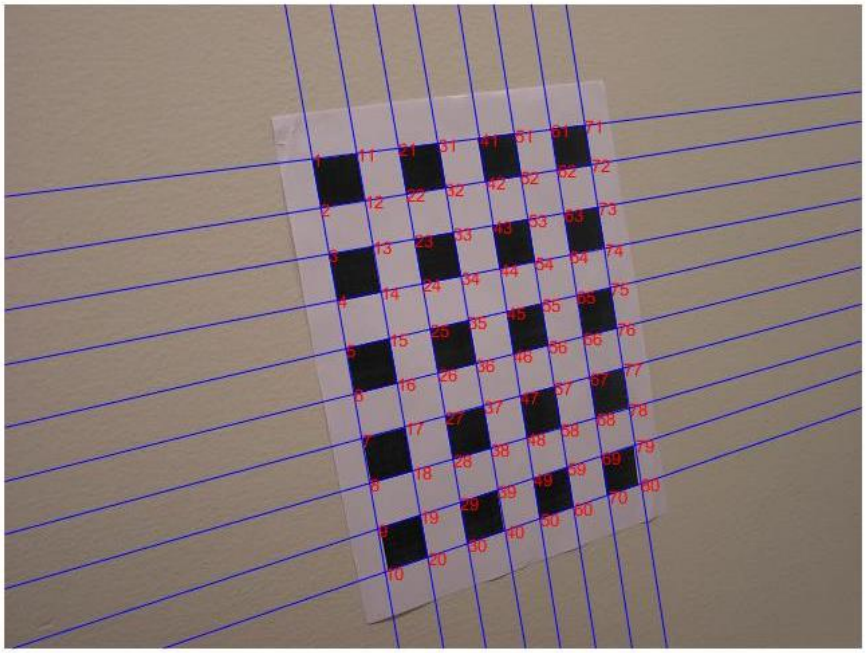
$$\left[R_{11}|\vec{t}_{11}\right]_{LM} = \begin{bmatrix} 0.9 & -0.0082 & 0.0418 & -78.9214 \\ 0.0084 & 0.9999 & -0.0568 & -95.9668 \\ -0.0418 & 0.0569 & 0.9998 & 630.0702 \end{bmatrix}$$



(a)

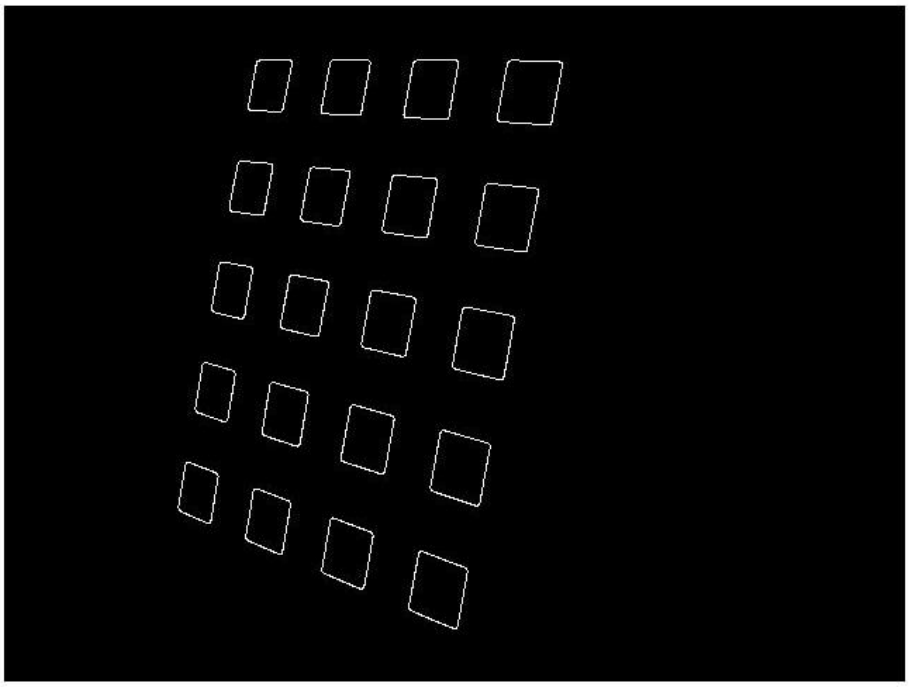


(b)

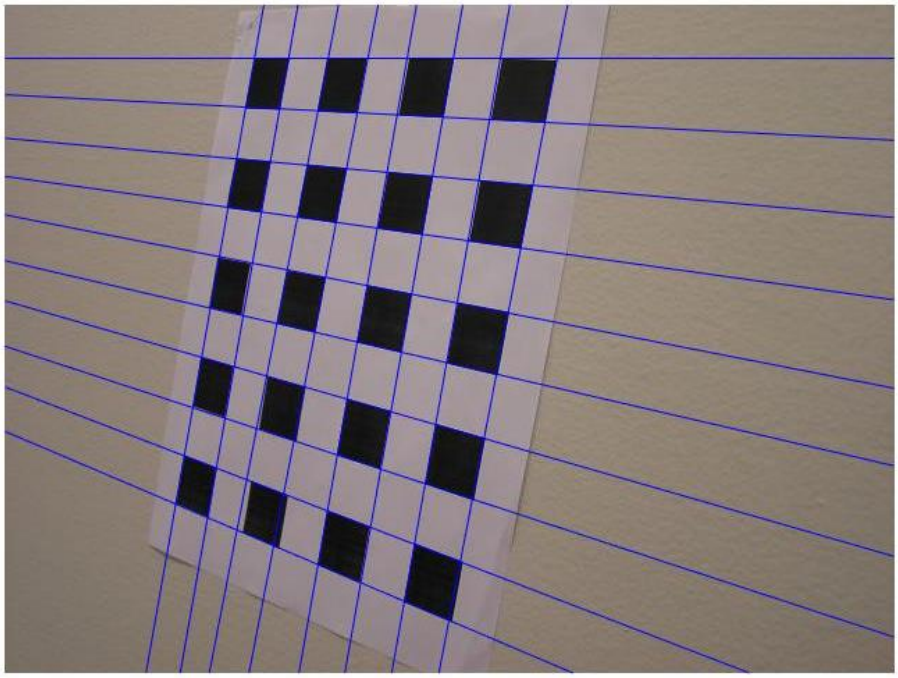


(c)

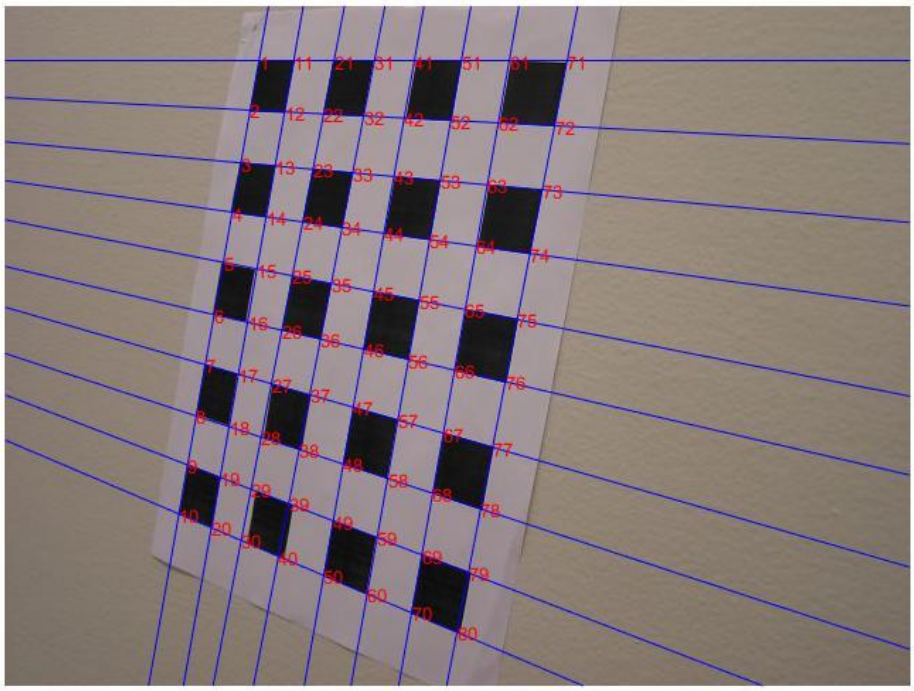
Fig 1: Edges, lines and corners of image 3, (a) edges, (b) Hough lines and (c) corners



(a)

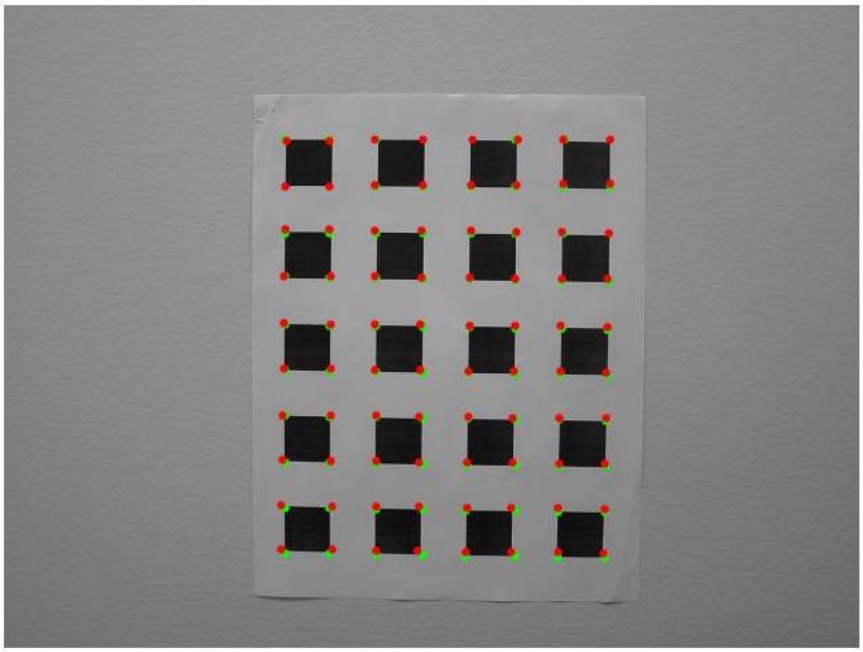


(b)

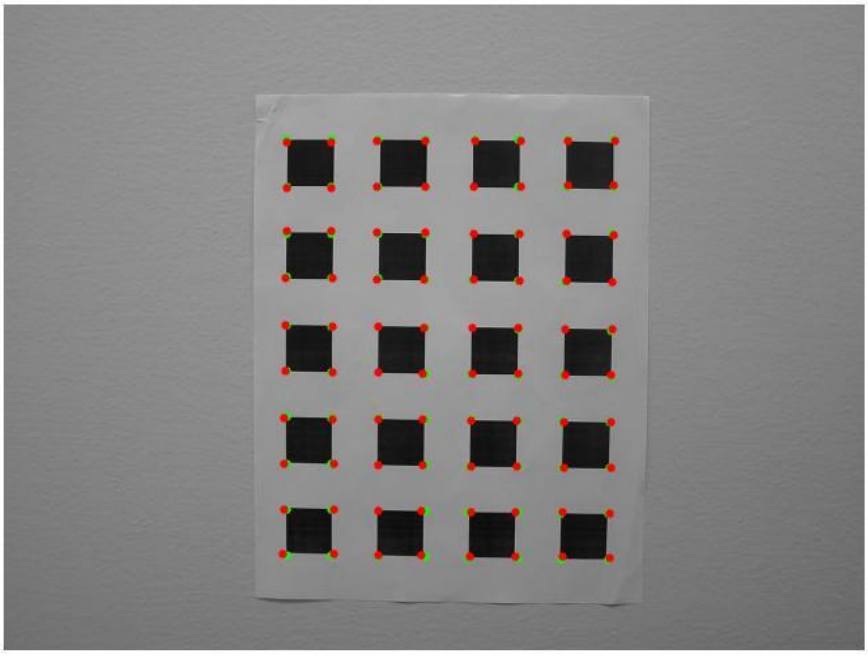


(c)

Fig 2: Edges, lines and corners of image 21, (a) edges (b) Hough lines and (c) corners

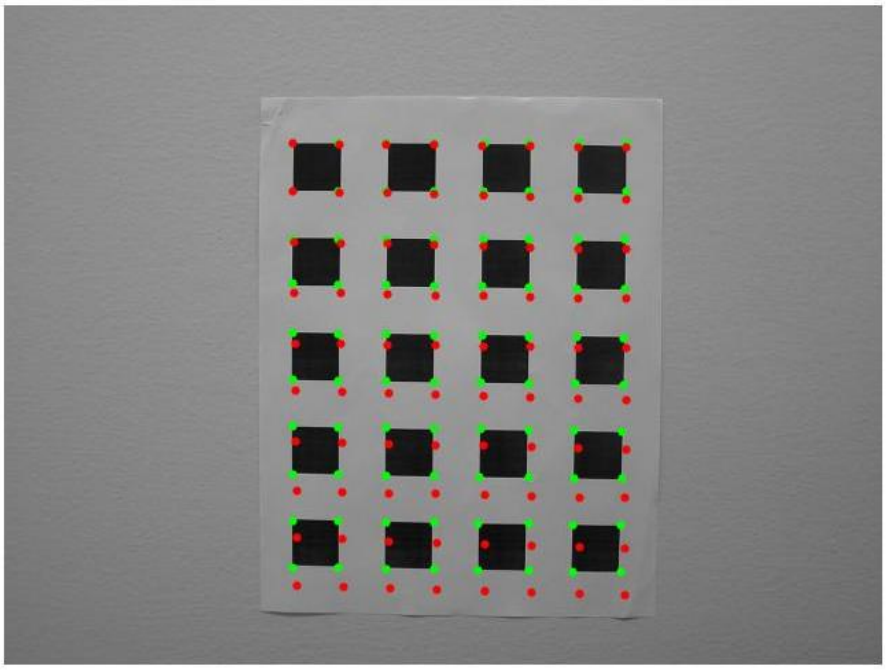


(a)

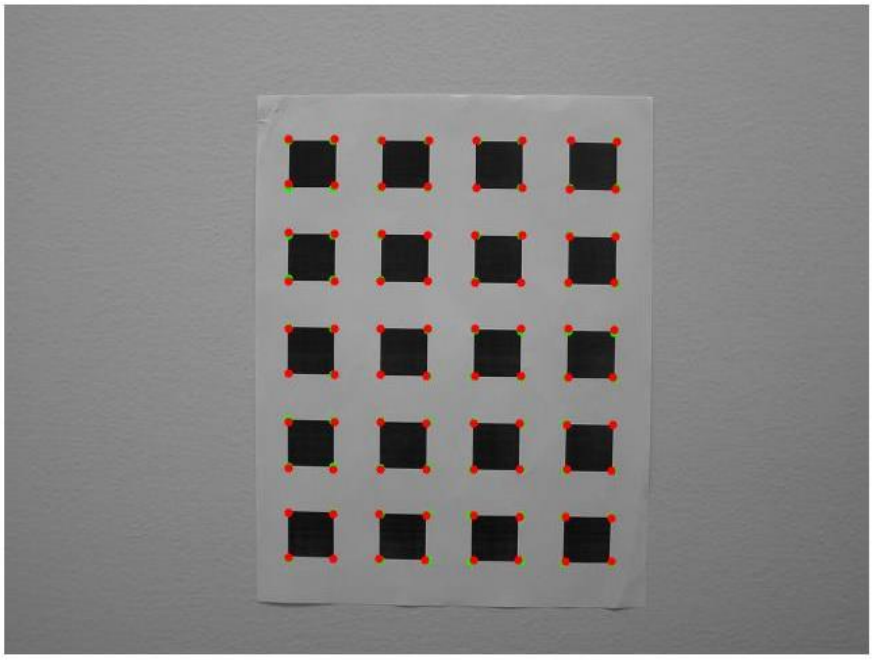


(b)

Fig 3: Reprojection of the corners in image 13 onto the fixed image (image 11). The green dots represent the actual corners, while the red represents the reprojections (a) using linear least squares (mean error = 1.1870 , variance = 0.8106) and (b) using LM refinement (mean error = 0.7293, variance = 0.3935). Labelling scheme is same as shown in the earlier images.

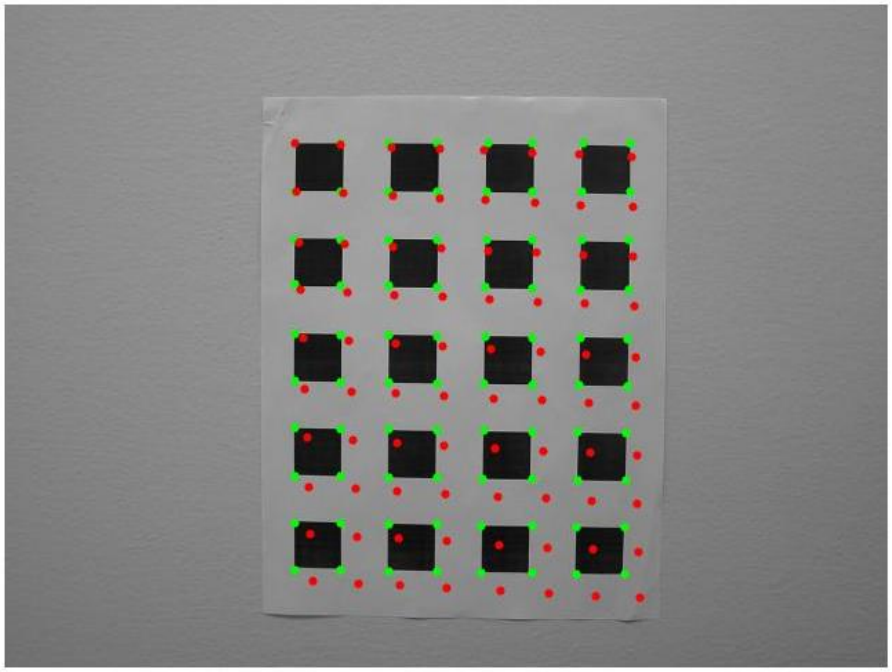


(a)

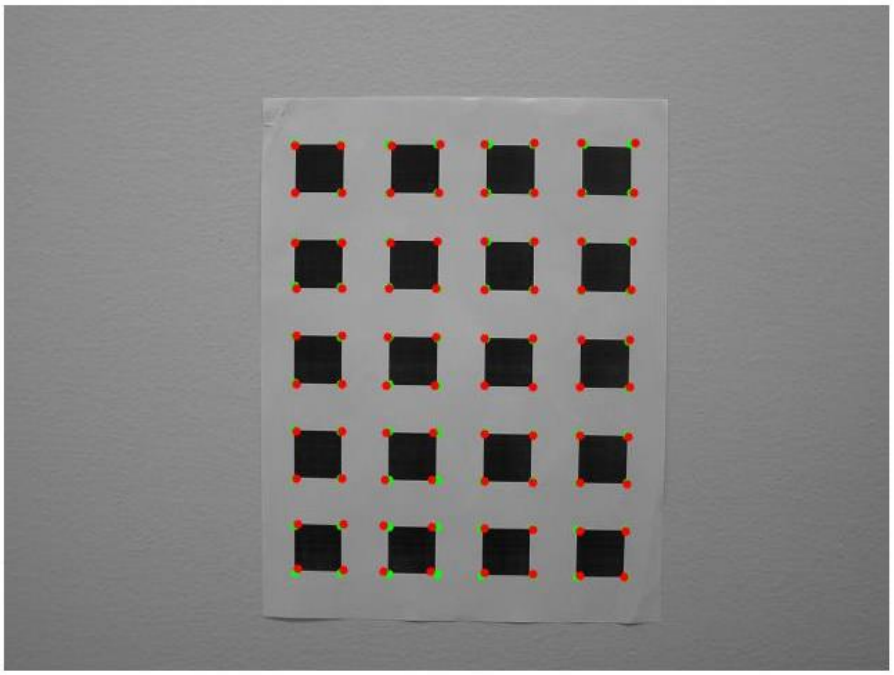


(b)

Fig 4: Reprojection of the corners in image 15 onto the fixed image (image 11). The green dots represent the actual corners, while the red represents the reprojections (a) using linear least squares (mean error = 5.2790 , variance = 24.8069) and (b) using LM refinement (mean error = 0.6704, variance = 0.2525). Labelling scheme is same as shown in the earlier images.

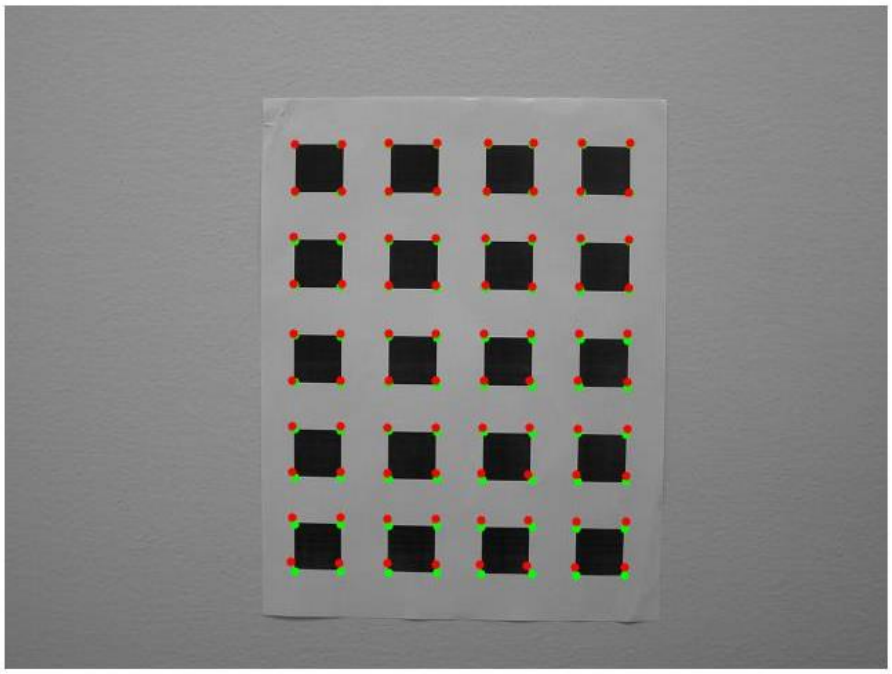


(a)

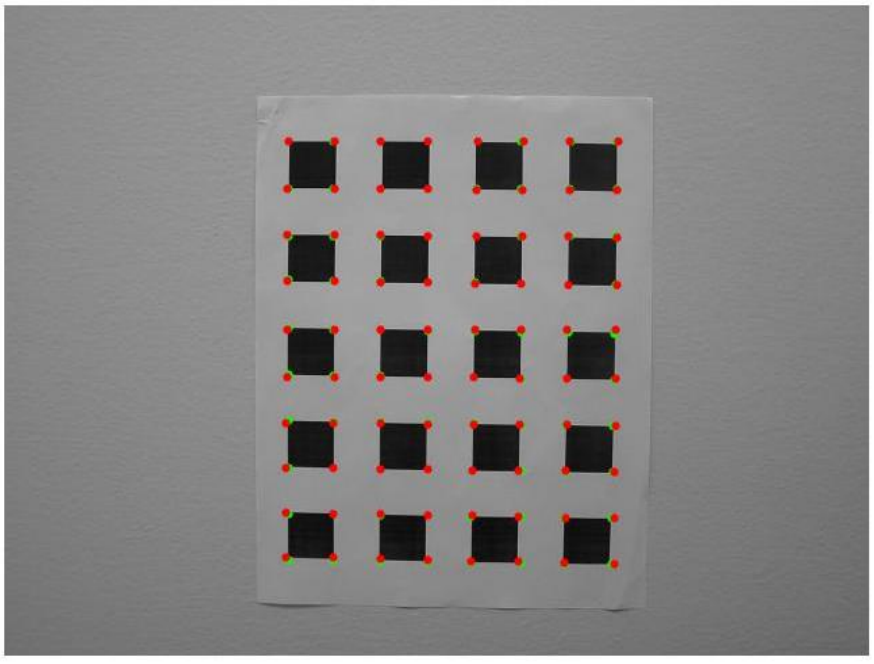


(b)

Fig 5: Reprojection of the corners in image 21 onto the fixed image (image 11). The green dots represent the actual corners, while the red represents the reprojections (a) using linear least squares (mean error = 7.6409 , variance = 18.7880) and (b) using LM refinement (mean error = 0.9854, variance = 0.8225). Labelling scheme is same as shown in the earlier images.



(a)



(b)

Fig 6: Reprojection of the corners in image 28 onto the fixed image (image 11). The green dots represent the actual corners, while the red represents the reprojections (a) using linear least squares (mean error = 1.9069 , variance = 3.0263) and (b) using LM refinement (mean error = 0.6323, variance = 0.3324). Labelling scheme is same as shown in the earlier images.

1.2.5. Quantitative Improvement in the performance due to LM refinement

Image Id	Mean LLS	Mean LM	Variance LLS	Variance LM
13	1.187018	0.729277	0.810649	0.393491
15	5.278965	0.670365	24.806902	0.252505
21	7.640938	0.985409	18.788027	0.822461
28	1.906926	0.632296	3.026308	0.332366

1.2.6. Quantitative Improvement in the performance due to LM refinement with radial distortion correction

A little to no improvement in the results were observed when the radial distortion was incorporated.

Image Id	Mean LLS	Mean LM	Variance LLS	Variance LM
13	1.187018	0.743997	0.810649	0.399554
15	5.278965	0.680425	24.806902	0.256226
28	1.906926	0.686515	3.026308	0.308637

1.3. Dataset 2:

1.3.1. Intrinsic parameters for the collected dataset:

(d) Linear least-squares (LLS):

$$K = \begin{bmatrix} 1209.4601 & 0.0218 & 628.6163 \\ 0 & 1206.5564 & 342.6448 \\ 0 & 0 & 1 \end{bmatrix}$$

(e) LM algorithm refined without radial distortion:

$$K = \begin{bmatrix} 1531.4326 & -7.1804 & 343.3914 \\ 0 & 1527.4570 & 616.9476 \\ 0 & 0 & 1 \end{bmatrix}$$

(f) LM algorithm refined with radial distortion:

k1 = 0.2680

k2 = -1.1899

$$K = \begin{bmatrix} 1512.0832 & -4.77628 & 335.065 \\ 0 & 1512.22412 & 572.5403 \\ 0 & 0 & 1 \end{bmatrix}$$

1.3.2. Extrinsic parameters for the collected dataset:

(e) Image 4 using LLS and LM:

$$[R_4|\vec{t}_4]_{LLS} = \begin{bmatrix} 0.9376 & -0.1146 & 0.3283 & -241.1180 \\ 0.1561 & 0.9824 & -0.1027 & -79.3356 \\ -0.3107 & 0.1476 & 0.9390 & 697.9133 \end{bmatrix}$$

$$[R_4|\vec{t}_4]_{LM} = \begin{bmatrix} 0.9338 & -0.1348 & 0.4595 & -83.6004 \\ 0.1832 & 0.9828 & -0.1644 & -256.3843 \\ -0.4425 & 0.2059 & 0.9311 & 952.3235 \end{bmatrix}$$

(f) Image 8 using LLS and LM:

$$[R_8|\vec{t}_8]_{LLS} = \begin{bmatrix} 0.9159 & -0.3826 & 0.1217 & -223.3248 \\ 0.3394 & 0.8993 & -0.1782 & -80.9940 \\ -0.0412 & 0.2118 & 0.9764 & 670.8773 \end{bmatrix}$$

$$[R_8|\vec{t}_8]_{LM} = \begin{bmatrix} 0.9490 & -0.3830 & 0.1631 & -68.5666 \\ 0.4068 & 0.9235 & -0.3039 & -241.5184 \\ -0.0884 & 0.3334 & 0.9652 & 879.2391 \end{bmatrix}$$

(g) Image 13 using LLS and LM:

$$[R_{13}|\vec{t}_{13}]_{LLS} = \begin{bmatrix} 0.9812 & -0.1332 & -0.1399 & -214.4210 \\ 0.0846 & 0.9473 & -0.3089 & -12.6848 \\ 0.1737 & 0.2912 & 0.9408 & 555.6498 \end{bmatrix}$$

$$[R_{13}|\vec{t}_{13}]_{LM} = \begin{bmatrix} 0.9866 & -0.0811 & -0.2073 & -81.0359 \\ 0.0300 & 0.9470 & -0.4402 & -134.0958 \\ 0.2205 & 0.4337 & 0.9353 & 674.3841 \end{bmatrix}$$

(h) Image 15 using LLS and LM:

$$[R_{15}|\vec{t}_{15}]_{LLS} = \begin{bmatrix} 0.8531 & 0.2470 & -0.4595 & -312.0590 \\ -0.3153 & 0.9459 & -0.0770 & -17.3559 \\ 0.4156 & 0.2106 & 0.8848 & 802.2033 \end{bmatrix}$$

$$[R_{15}|\vec{t}_{15}]_{LM} = \begin{bmatrix} 0.8593 & 0.3101 & -0.5220 & -112.2389 \\ -0.3760 & 0.9435 & -0.1062 & -181.6259 \\ 0.4767 & 0.2376 & 0.8931 & 924.7718 \end{bmatrix}$$

1.3.3. Extrinsic parameters with LM and distortion incorporated:

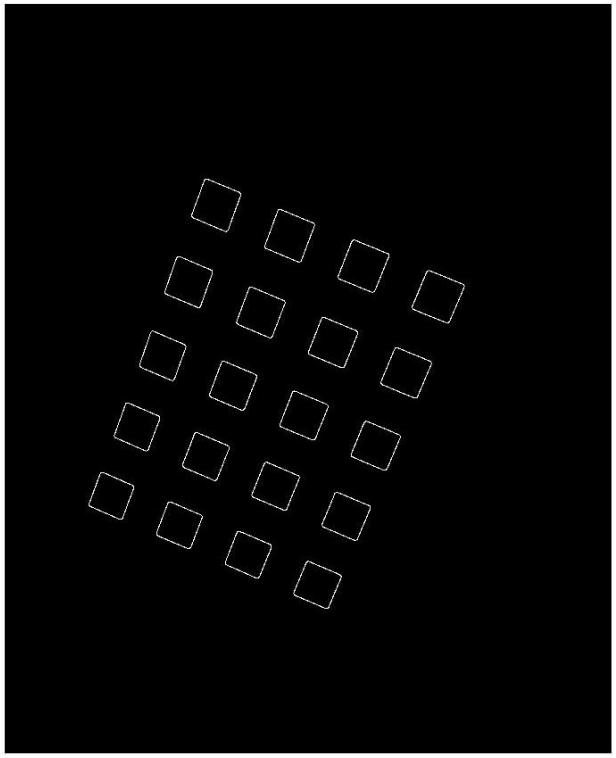
$$\left[R_{15}|\vec{t}_{15}\right]_{LM} = \begin{bmatrix} 0.8653 & 0.3110 & -0.5160 & -107.2044 \\ -0.3619 & 0.9509 & -0.0746 & -154.8972 \\ 0.4817 & 0.1995 & 0.9004 & 923.2356 \end{bmatrix}$$

1.3.4. Camera pose for fixed imaged with measured ground-truth :

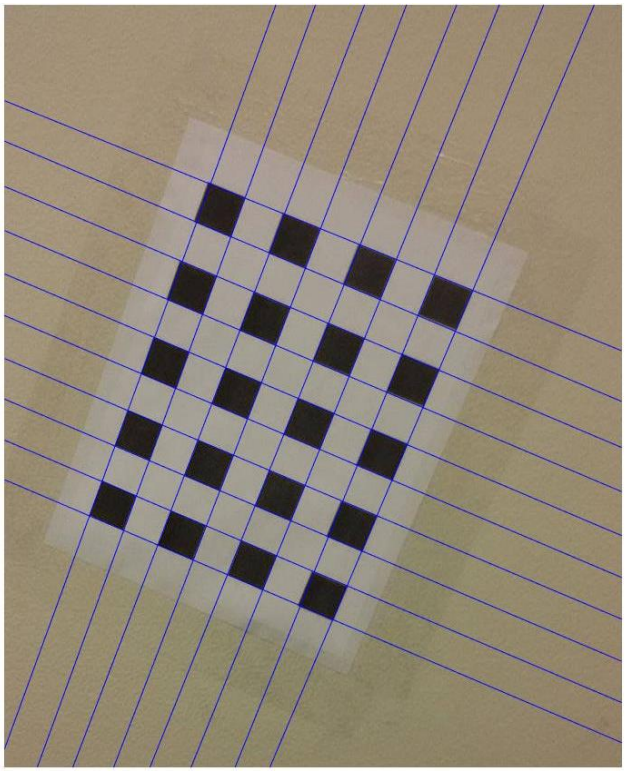
The $\left[R_1|\vec{t}_1\right]$for the fixed image 1 with LLS and LM are given below. It can be seen that R is approximately identity matrix with t representing the translation.

$$\left[R_1|\vec{t}_1\right]_{LLS} = \begin{bmatrix} 0.9995 & -0.0299 & -0.003 & -266.9536 \\ 0.0293 & 0.9931 & -0.1139 & -30.1586 \\ 0.0067 & 0.1137 & 0.9935 & 705.1718 \end{bmatrix}$$

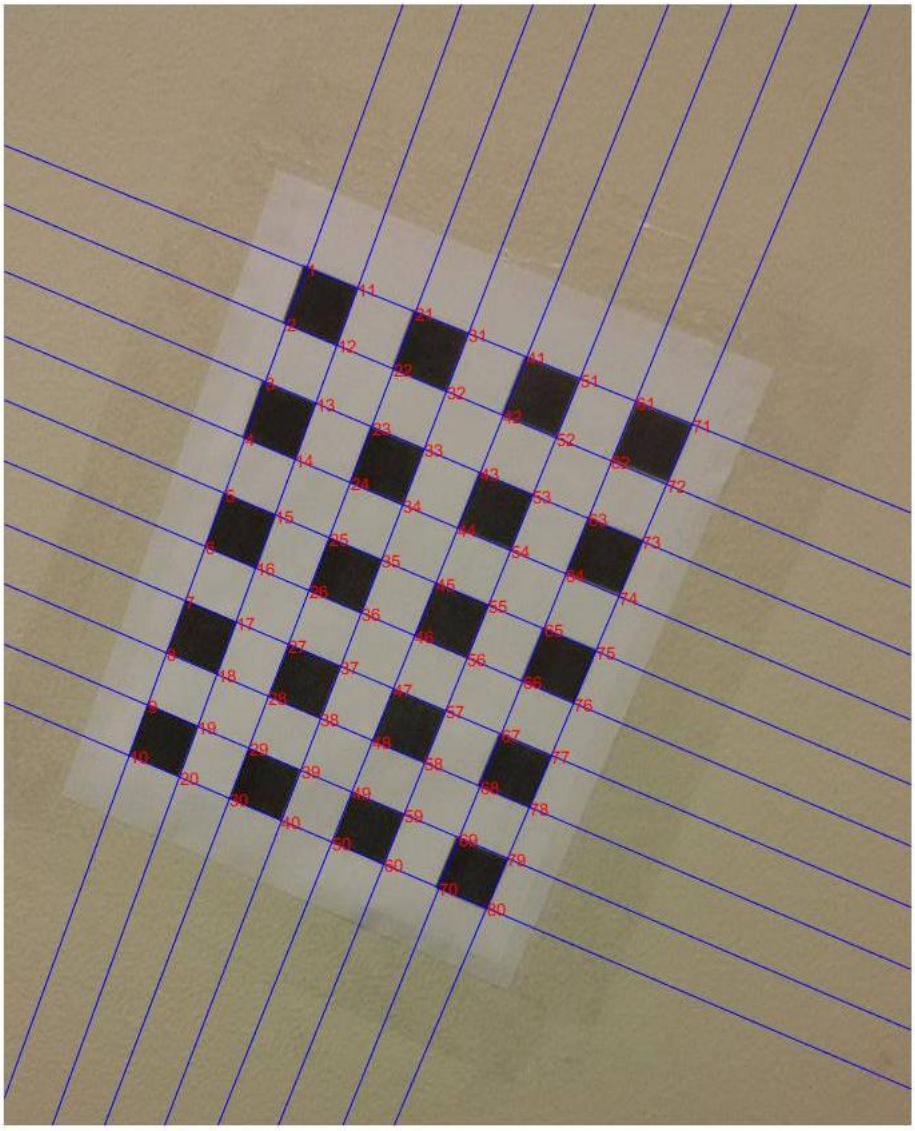
$$\left[R_1|\vec{t}_1\right]_{LM} = \begin{bmatrix} 0.9999 & -0.0121 & -0.0301 & -101.1858 \\ 0.0114 & 0.9985 & -0.1434 & -190.6405 \\ 0.0304 & 0.1434 & 0.9984 & 892.4544 \end{bmatrix}$$



(a)

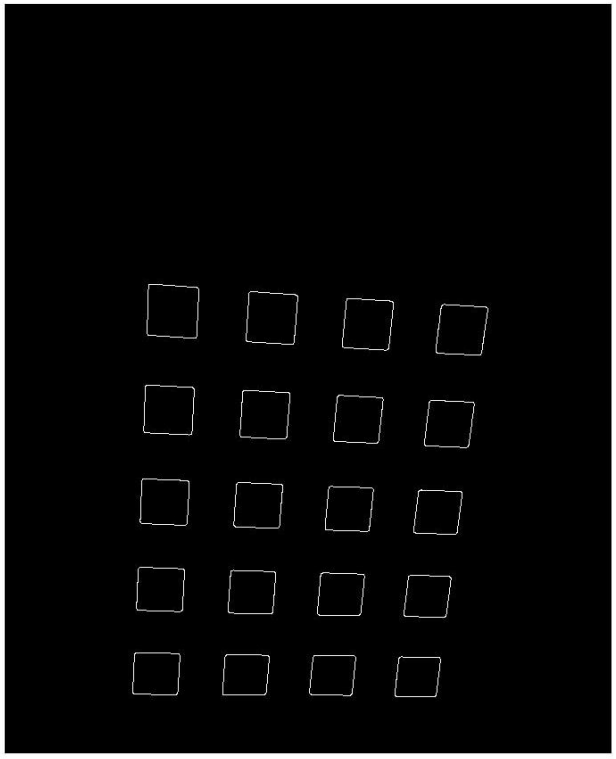


(b)

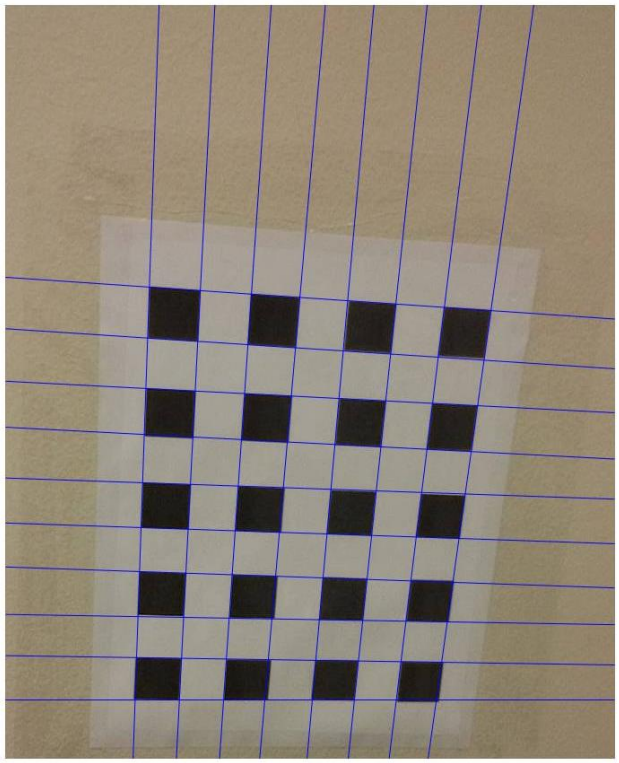


(c)

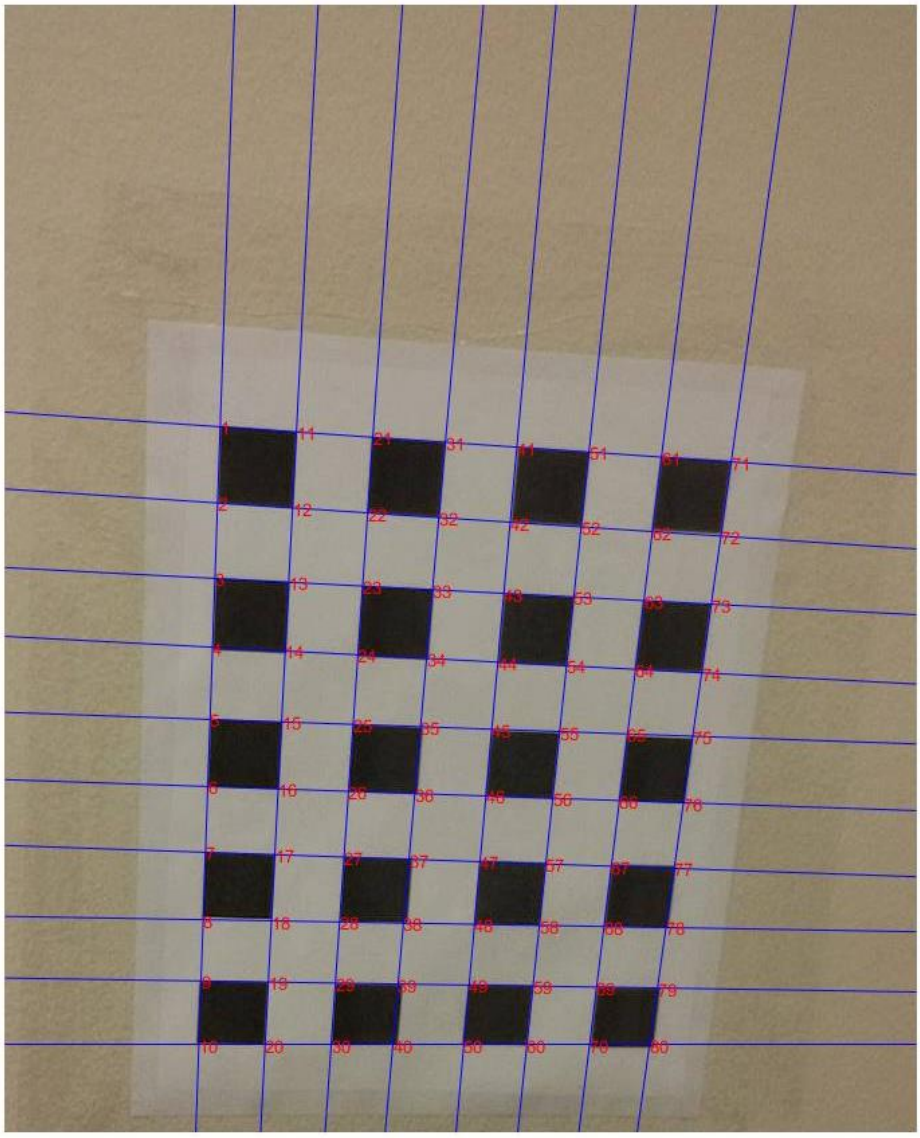
Fig 1: Edges, lines and corners of image 8, (a) edges, (b) Hough lines and (c) corners



(a)

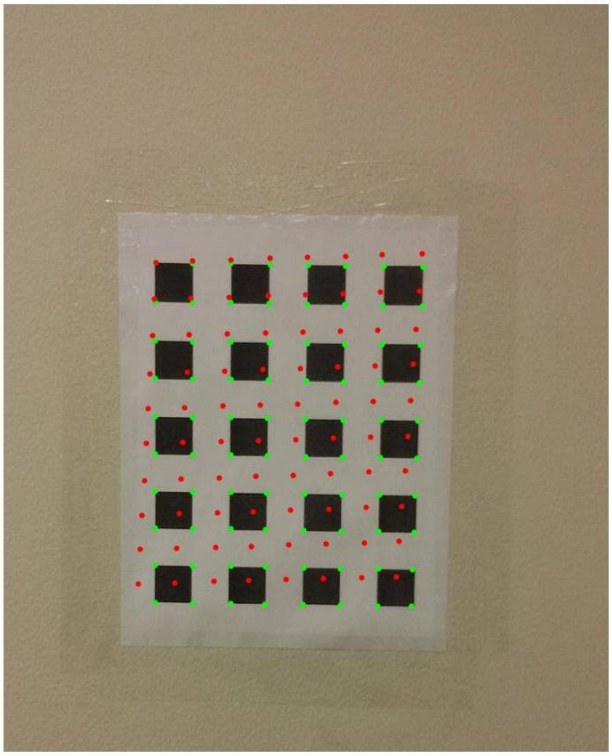


(b)

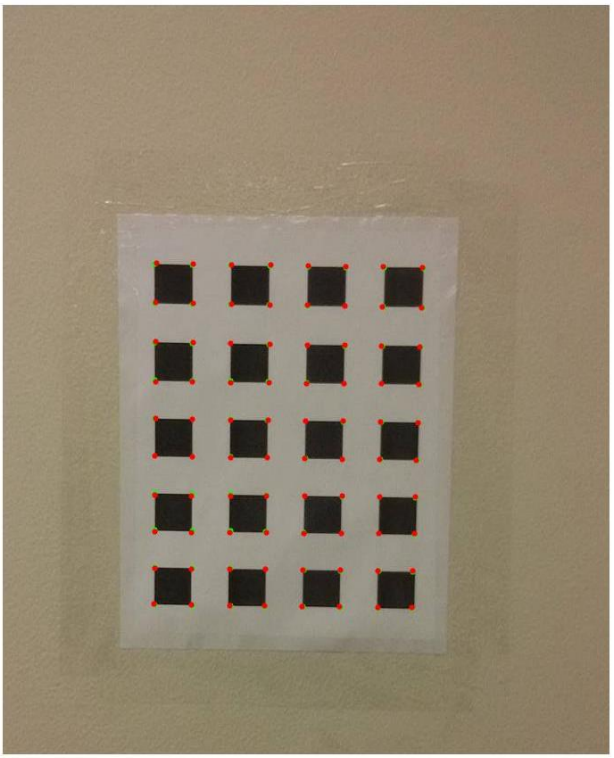


(c)

Fig 2: Edges, lines and corners of image 13, (a) edges (b) Hough lines and (c) corners

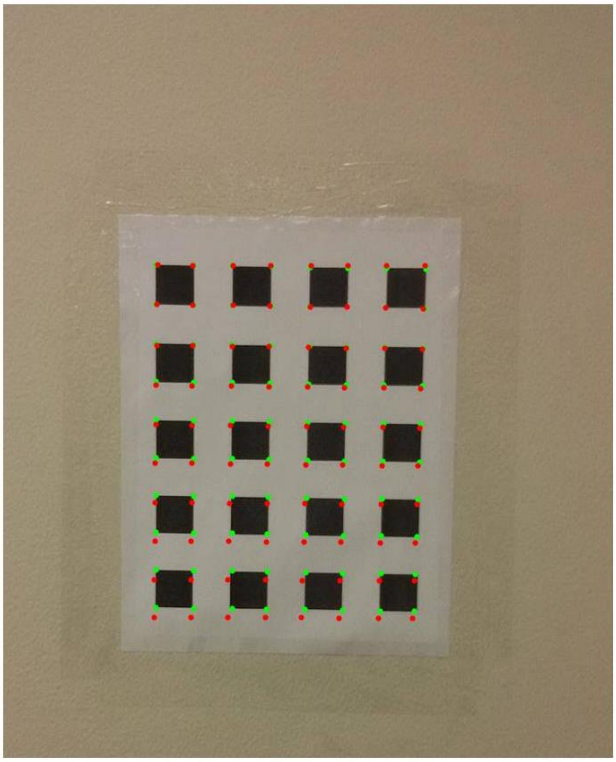


(a)

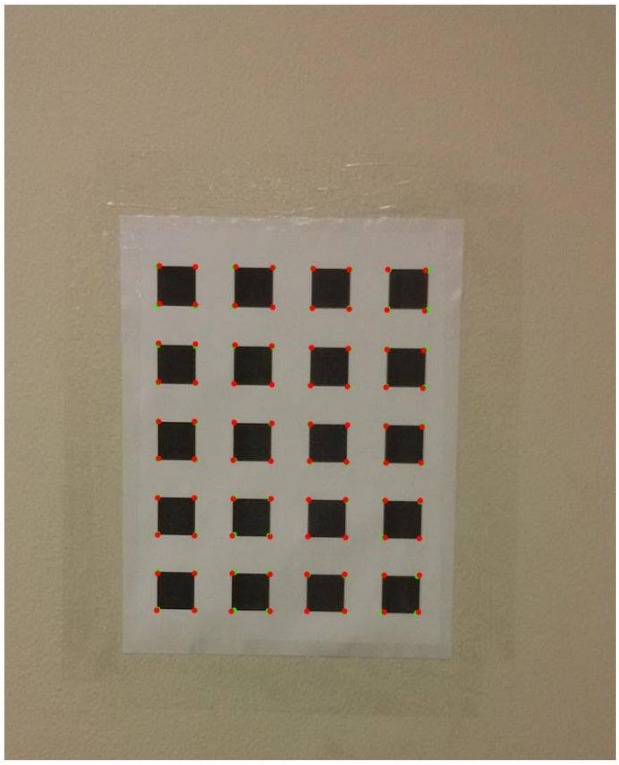


(b)

Fig 3: Reprojection of the corners in image 4 onto the fixed image (image 1). The green dots represent the actual corners, while the red represents the reprojections (a) using linear least squares (mean error = 13.2835 , variance = 60.9720) and (b) using LM refinement (mean error = 0.5716, variance = 0.2195). Labelling scheme is same as shown in the earlier images.

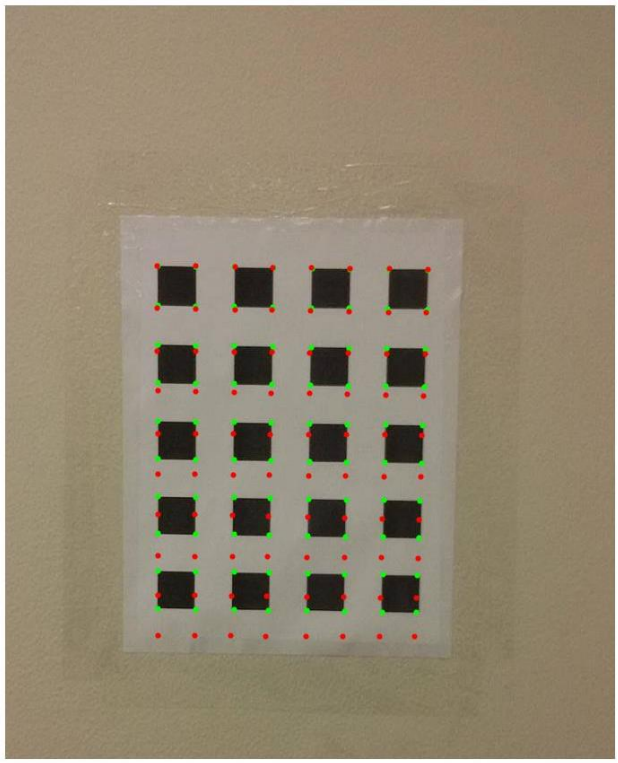


(a)

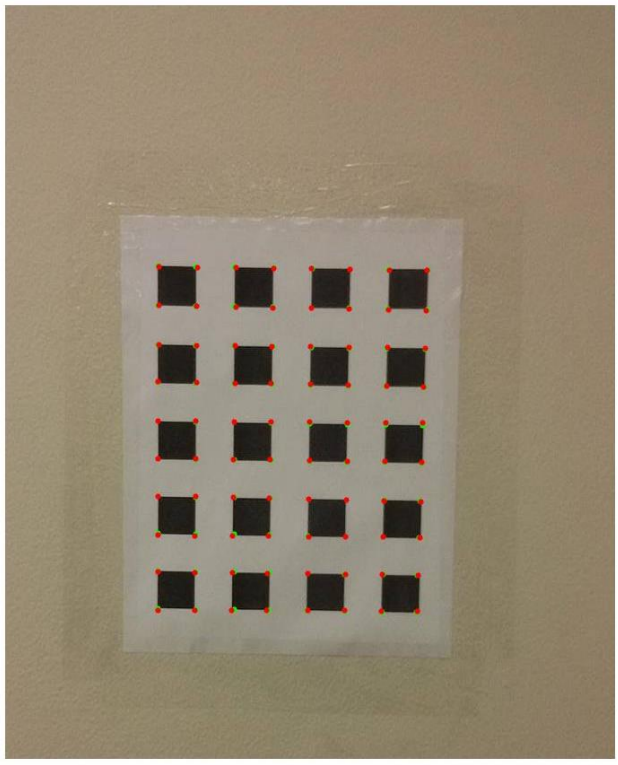


(b)

Fig 4: Reprojection of the corners in image 8 onto the fixed image (image 1). The green dots represent the actual corners, while the red represents the reprojections (a) using linear least squares (mean error = 3.2621 , variance = 8.6140) and (b) using LM refinement (mean error = 0.6930, variance = 0.3519). Labelling scheme is same as shown in the earlier images.

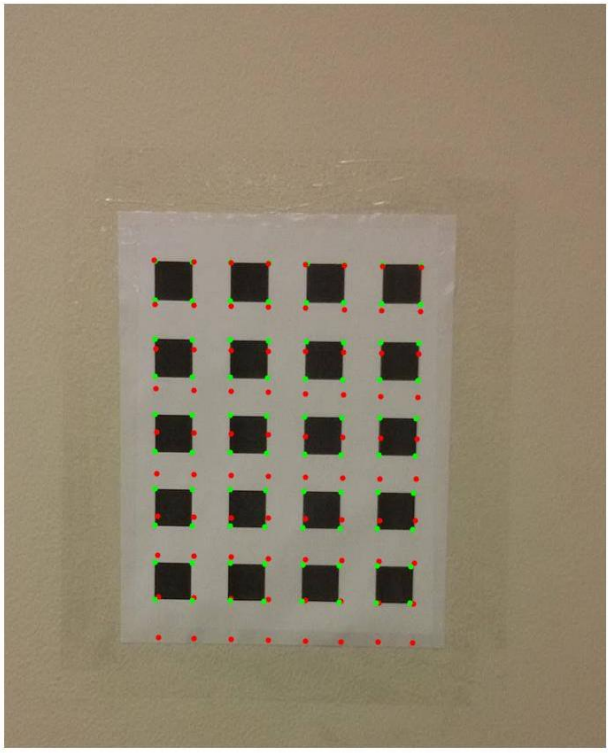


(a)

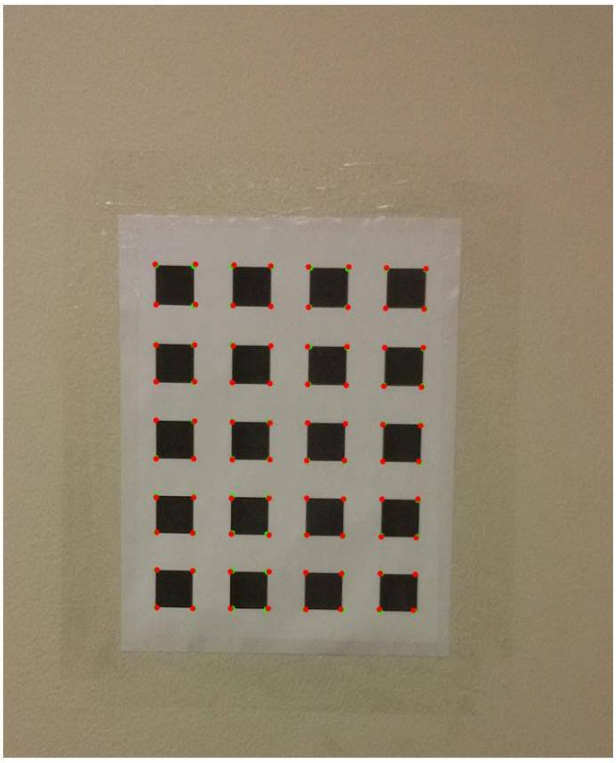


(b)

Fig 5: Reprojection of the corners in image 13 onto the fixed image (image 1). The green dots represent the actual corners, while the red represents the reprojections (a) using linear least squares (mean error = 7.8650 , variance = 91.0762) and (b) using LM refinement (mean error = 0.7018, variance = 0.3241). Labelling scheme is same as shown in the earlier images.



(a)



(b)

Fig 6: Reprojection of the corners in image 15 onto the fixed image (image 1). The green dots represent the actual corners, while the red represents the reprojections (a) using linear least squares (mean error = 12.5323, variance = 217.4984) and (b) using LM refinement (mean error = 0.6664, variance = 0.2656). Labelling scheme is same as shown in the earlier images.

1.3.5. Quantitative Improvement in the performance due to LM refinement

Image Id	Mean LLS	Mean LM	Variance LLS	Variance LM
4	13.28349	0.57162	60.97204	0.21949
8	3.26210	0.69301	8.61401	0.35190
13	7.86501	0.70177	91.07615	0.32407
15	12.53229	0.66636	217.49842	0.26559

1.3.6. Quantitative Improvement in the performance due to LM refinement with radial distortion correction

No improvement in the results were observed when the radial distortion was incorporated. Including the radial distortion resulted in increased mean and variance of errors compared to the just using LM refinement.

Image Id	Mean LLS	Mean LM	Variance LLS	Variance LM
4	13.28349	0.75518	60.97204	0.49506
8	3.26210	0.76785	8.61401	0.36730
13	7.86501	0.76927	91.07615	0.37534
15	12.53229	0.72628	217.49842	0.34821

1.4. References:

1) Zhengyou Zhang. A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22:1330-1334, December 2000. MSR-TR-98-71, Updated March 25, 1999.

3. Source Code:

3.1.Function calls for Task 1.1:

```
'''%----------- This is main fuunction--------%
close all; clc; clear all;
%Flag to indicate whether distortion needs to be included in the model or
%not.
distort_flag = 1;
box_size = 25; % blocksize mm

%Parameters of dataset1
n_images = 40;
direc = '..\Files\Dataset1\';
h_res = 0.6; can_thresh = 0.8;
standard_img = 11;
backproj_imgs = [13, 15, 21, 28];

%Parameters of dataset2
```

```
% n_images = 20;
% direc = '..\Files\Dataset2';
% h_res = 0.7; can_thresh = 0.7;
% standard_img = 1;
% backproj_imgs = [4, 8, 13, 15];

% Get the world coordinates of the checkerboard pattern using the box
% dimensions physically measured in mm
coords_world = find_world_coords(box_size);
%Compute the corners, homographies and V matrices for all images
[H_Dataset,Corners_Dataset,V_Dataset] =
ComputeDatasetProperties(n_images,direc,h_res,can_thresh,coords_world);
% Find the intrinsic parameters of the camera using linear least squares
[U,D,V] = svd(V_Dataset);
b = V(:,6);
K = Find_K(b);

% Find the Extrinsic parameters of images using linear least squares
R_LLs = cell(1,n_images);
t_LLs = cell(1,n_images);
intial_estimates_vec = zeros(1,n_images*6);
for k = 1:n_images
    H = H_Dataset{k};
    [R,t] = Find_Extrinsic(H,K);
    R_LLs{k} = R;
    t_LLs{k} = t;
    w = R_Rodriguez(R);
    intial_estimates_vec(1,(k-1)*6+1:k*6) = [w',t'];
end
%Add the initial values of k1 and k2 if distortion needs to be corrected
if (~distort_flag)
    intial_estimates_vec =[intial_estimates_vec,K(1,1:3),K(2,2:3)];
else
    intial_estimates_vec =[intial_estimates_vec,K(1,1:3),K(2,2:3),0,0];
end

%Optimize the intrisic and extrinsic parameters using Levnberg algorithm
opts = optimoptions('lsqcurvefit','Algorithm','levenberg-marquardt');
final_estimates_vec =
lsqnonlin(@func_LM_cost,intial_estimates_vec,[],[],opts,coords_world,Corners_
Dataset,distort_flag,n_images);

%Optimized intrisic parameters
K1 = [final_estimates_vec(1,6*n_images+1:6*n_images+3);
0,final_estimates_vec(1,6*n_images+4:6*n_images+5); 0, 0, 1];
if(distort_flag)
    k1 = final_estimates_vec(1,6*n_images+6);
    k2 = final_estimates_vec(1,6*n_images+7);
end

%Get Extrinsic Parameters using optimized estimates
R_LM = cell(1,n_images);
t_LM = cell(1,n_images);
for k = 1:n_images
     wt = final_estimates_vec(1,(k-1)*6+1:k*6);
     omega = wt(1,1:3);
```

```
     t_LM{k} = wt(1,4:6)';
     R_LM{k} = Rodriguez_R(omega);
end
%Initialize the stats for checking the backprojection quality
residual_LL = zeros (length(backproj_imgs),2);
residual_LM = zeros (length(backproj_imgs),2);
for i = 1:length(backproj_imgs)
    residual_LL(i,:) =
ReprojectCorners(R_LLs,t_LLs,K,Corners_Dataset,direc,standard_img,backproj_im
gs(i));
    residual_LM(i,:) =
ReprojectCorners(R_LM,t_LM,K1,Corners_Dataset,direc,standard_img,backproj_img
s(i));
end
```

A. Different functions for Camera calibration using Zhang's algorithm with LM refinement:

```
function [H_Dataset,Corners_Dataset,V_Dataset] =
ComputeDatasetProperties(n_images,direc,h_res,can_thresh,world_coord)
%Store the corners,homographies and V matrices for the entire dataset
Corners_Dataset = cell(1,n_images);
H_Dataset = cell(1,n_images);
V_Dataset = zeros(n_images*2,6);

for i = 1:n_images
    filename = strcat([direc, '\Pic_' num2str(i), '.jpg']);
    %Find and plot the coners of images
    Corners_Dataset{i} = FindandPlotCorners(filename,h_res,can_thresh);
    % Check if the corners are not 80 and report as warning
    if length(Corners_Dataset{i}) ~= 80
        fprintf('Number of corners not detected correctly for image number %d
\n',i);
    end
    % Find the homography between the image and world coordinates
    H_Dataset{i} = find_homography(world_coord,Corners_Dataset{i});
    %Find the V (2*6) matrix for every images
    V = Find_V_Matrix(H_Dataset{i});
    V_Dataset((i-1)*2+1:i*2,:) = [V(2,:);(V(1,:)-V(3,:))];
end
end
```

```
function A = A_Matrix_Generator(domain_point,range_point)
%function for generating a 2 x 9 design matrix needed to compute Homography
%Inputs:
%   domain_point: Coordinates of a point in the domain image (x,y)
%   range_point: Coordinates of corresponding point in the range image
(x',y')
%Output:
%   A: A 2 x 9 design matrix
x =domain_point(1,1); y=domain_point(1,2);
xr =range_point(1,1); yr=range_point(1,2);

A = [0,0,0,-x,-y,-1,yr*x,yr*y,yr ; x,y,1,0,0,0,-xr*x,-xr*y,-xr];
```

```
end
function [R,t] = Find_Extrinsic(H,K)
% Get the columns of the homography matrix
h1 = H(:,1);
h2 = H(:,2);
h3 = H(:,3);
%Get the K inverse
K_inv = pinv(K);
% Compute the scaling factor from the orthonormality condition of R
scale = 1/norm(K_inv*h1);
% Get the unscaled t vector
t = K_inv * h3;
% If 3 element of t is negative scale should be negative
if t(3) < 0
    scale = -scale;
end
%Find the scaled elements of rotation matrix for the image under
consideration
r1 = scale*K_inv*h1;
r2 = scale*K_inv*h2;
r3 = cross(r1,r2);
R = [r1, r2, r3];
%Condition the R matrix
R = Condition_R(R);
% Scale the t vector
t = scale * t;

end

function R=Condition_R(R)
    %Decompose R and reset all singular values to 1
    [U,D,V] = svd(R);
     R = U * V';
end

function H = find_homography(Domain_pts,Range_pts)
%function for estimating the 3 x 3 Homography matrix---Modified from HW2
%Inputs:
%   Domain_pts: An n x 2 array containing coordinates of domian image
points(Xi,Yi)
%   range_point: An n x 2 array containing coordinates of range image
points(Xi',Yi')
%Output: A 3 x 3 Homography matrix

%Find num of points provided
n = length(Domain_pts);
%Initialize A Design matrix having size of 2n x 8
A = zeros(2*n,9);
%Loop through all the points provided and stack them vertically, this will
result in 2n x 9 Design matrix
for i=1:n
    A((i-1)*2+1:i*2,:)=A_Matrix_Generator(Domain_pts(i,:),Range_pts(i,:));
end
%Decompose A using SVD decomposition
[U,D,V] = svd(A);
h = V(:,9); %Eigen vector corresponding to the smallest eigen value of D
```

```
% Rearrange the vector h to Homography matrix H
H = [h(1:3)'; h(4:6)'; h(7:9)'];
end

function K = Find_K(b)
w11 = b(1);
w12 = b(2);
w22 = b(3);
w13 = b(4);
w23 = b(5);
w33 = b(6);
% Find the Intrinsic parameters
x0 = (w12* w13- w11* w23)/(w11 * w22 - w12^2);
lambda = w33-(w13^2 + x0 *(w12 * w13 - w11 * w23))/w11;
alphaX = sqrt(lambda / w11);
alphaY = sqrt(lambda * w11/(w11*w22 - w12^2));
s = - w12 * alphaX^2 * alphaY/lambda;
y0 = s*x0/alphaY - (w13 * alphaX^2/lambda);
K = [alphaX, s, x0; 0, alphaY, y0; 0, 0, 1];

 end

function V = Find_V_Matrix(H)
%Function for finding the Vij from Homography matrix.
%Inputs:
%   H: Homography matrix between world and pixel coordinates
%Outputs:
%   V: A matrix containing the Vij. V11,V12 and V22 are at 1st,2nd and 3rd
%   row, respectively.
ij = [1,1;1,2;2,2];
V = zeros(3,6);
for k=1:length(ij)
    i = ij(k,1);
    j = ij(k,2);
    V(k,:) = [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)*H(2,j)+H(2,i)*H(3,j),H(3,i)*H(3,j)];
end
end

function coords_world = find_world_coords(board_size)
%Intialize the world coordinates
coords_world = zeros(80,2);
%Generate World Coordinates
for i = 1:8
    for j = 1:10
        coords_world((i-1)* 10 + j,:)=[(i-1)*board_size, (j-1)*board_size];
    end
end
end
function Imagecorners_sorted = FindandPlotCorners(filename,h_res,can_thresh)
% Read the color image,convert to gray scale and perform canny edge
% detection
```

```
ColorImg = imread(filename);
gray_img = rgb2gray(ColorImg);
edges = edge(gray_img,'canny',can_thresh);
% Show the images with edges
figure()
imshow(edges)

%--- Start to perform hough transformation----%
[H, theta, rho] = hough(edges,'RhoResolution',h_res);

% There are 5 pairs of horizontal and 4 pairs of verticle lines, so total
% 18 lines. These are 18 strongest lines.
peaks = houghpeaks(H,18,'threshold',ceil(0.2*max(H(:))));
% Pick the bigger fillgap value otherwise many small lines will appear.
% Discard some small lines
lines = houghlines(edges,theta,rho,peaks,'FillGap',150,'MinLength',70);

%Plot the lines
figure()
imshow(ColorImg)
hold on
for k = 1:length(lines)
    pt1 = lines(k).point1;
    pt2 = lines(k).point2;
    % Find the vertical lines and plot them along the entire image
    if abs(lines(k).theta) <=1 %Allowence for lines whose theta is not
exactly zero
        y = 1:size(gray_img,1);
        x = pt1(1,1)*ones(1,length(y));
        plot(x,y,'Color','blue');
    % Find the horizontal lines and plot them along the entire image.
    %Can't fix their position as boxes are inclined in x direction for
multiple images.
    else
        % Find the line parameters
        m = (pt1(1,2)-pt2(1,2))/(pt1(1,1)-pt2(1,1));
        c = pt1(1,2) - m*pt1(1,1);
        x = 1:size(gray_img,2);
        y =  m*x + c;
        plot(x,y,'Color','blue');
    end
end

%Find the corners and plot them
Imagecorners_sorted = FindCorners(lines,gray_img);

for i = 1:length(Imagecorners_sorted)
    hold on

text(Imagecorners_sorted(i,1),Imagecorners_sorted(i,2),int2str(i),'Color','r'
);
end
hold off;
end

function sorted_corners = FindCorners(lines,gray_img)
```

```
%Function for finding and sorting the corners
%Divide the lines in horizontal vs vertical lines
mask = abs([lines.theta])<=40;
vertical_lines = lines(mask);
horizontal_lines = lines(~mask);
%Initialize an array of corners
corners = zeros(length(vertical_lines)*length(horizontal_lines),2);

for i= 1:length(vertical_lines)
    for j = 1:length(horizontal_lines)  %For every vertical lines check all
horizontal lines
        %Find the intersection between every pair of horizontal and
        %vertical lines
        intersection = IntersectLines(vertical_lines(i),
horizontal_lines(j));
        %Check if the intersection is within image bounds
        if(intersection(1)>1 && intersection(2)>1 &&
intersection(1)<size(gray_img,2) && intersection(2)<size(gray_img,1))
            %Append the nrew corners point in the list
            corners ((i-1)*10+j,:) = intersection;
        end
    end
end

%Find the minimum corners along eight vertical lines
x = zeros(length(vertical_lines),1);
for i=1:length(vertical_lines)
x(i)=min(corners((i-1)*10+1:i*10));
end
[~, ind] = sort(x,'ascend');

%Sort the eight vertical lines and their corresponding horizontal corners
xs10 = zeros(length(vertical_lines)*length(horizontal_lines),2);
for i=1:8
    xs10((i-1)*10+1:i*10,:)=corners((ind(i)-1)*10+1:(ind(i)-1)*10+10,:);
end

%Finally sort the 10 corners for every vertical lines based on the y values
sorted_corners = zeros(80,2);
for i=1:8
    Imagecorners_temp = xs10((i-1)*10+1:i*10,:);
    sorted_corners((i-1)*10+1:i*10,:) =
sortrows(Imagecorners_temp,2,'ascend');
end
end

function error =
func_LM_cost(intial_estimates_vec,worldCoord,pixelCoord,dist_falg,num_imgs)
%Get initial values of intrinsic parameters
K = [intial_estimates_vec(6*num_imgs+1:6*num_imgs+3); 0,
intial_estimates_vec(6*num_imgs+4:6*num_imgs+5); 0, 0, 1];
% Find k1 and k2 if the radial distortion is needed
if(dist_falg == 1)
    k1 = intial_estimates_vec(6*num_imgs+6);
    k2 = intial_estimates_vec(6*num_imgs+7);
```

```
    K1 = [intial_estimates_vec(6*num_imgs+1), 0,
intial_estimates_vec(6*num_imgs+3); 0,
intial_estimates_vec(6*num_imgs+4:6*num_imgs+5); 0 0 1];
end

error = zeros(num_imgs*160,1);
%loop through all the images
for k = 1:num_imgs
     wt = intial_estimates_vec(1,(k-1)*6+1:k*6);
     omega = wt(1,1:3);
     t = wt(1,4:6)';
     R = Rodriguez_R(omega);
     projPixelCoord = zeros(80,2);
     %loop throough all the points in an images
     for i = 1:80
         x = K * [R t] *[worldCoord(i,:), 0, 1]';
         projPixelCoord(i,:) = [x(1)/x(3), x(2)/x(3)];
         %Correct the radial distortion of all the points if it is checked
         if(dist_falg == 1)
            xp = [projPixelCoord(i,:), 1];
            x_hat = pinv(K1)*xp';
            r2 = x_hat(1)^2 + x_hat(2)^2;
            x_rad = x_hat(1) + x_hat(1)*(k1*r2+k2*r2^2);
            y_rad = x_hat(2) + x_hat(2)*(k1*r2+k2*r2^2);
            x = K1*[x_rad y_rad 1]';
            projPixelCoord(i,:) = [x(1)/x(3) x(2)/x(3)];
         end
     end
     e = pixelCoord{k}-projPixelCoord;
     error((k-1)*160+1:k*160) = [e(:,1);e(:,2)];
end
end

function intersection = IntersectLines(lineA, lineB)
%% Code converted from HW1
%Inputs:
%   lineA and lineB: a structure containing the XY coordinates of two points
%          that can define the lines using crossproduct of points
%Output
%   intersection: An XY representation of the Intersection point
%%
%First from the two pints in HC
lineA = cross([lineA.point1,1], [lineA.point2, 1]);
%Second line from the two points in HC
lineB = cross([lineB.point1, 1],[lineB.point2, 1]);
intersection = cross(lineA, lineB);
% Change the HC to XY coordinates
intersection = double([intersection(1)/intersection(end)
intersection(2)/intersection(end)]);
end

function omega =R_Rodriguez(R)
phi = acos((trace(R)-1)/2);
omega = phi/(2*sin(phi))*([R(3,2)-R(2,3), R(1,3)-R(3,1), R(2,1)-R(1,2)])';
end
```

```
function R =Rodriguez_R(omega)
wx = [0 -omega(3) omega(2); omega(3) 0 -omega(1); -omega(2) omega(1) 0];
phi = norm(omega);
R = eye(3)+ sin(phi)/phi * wx + (1-cos(phi))/phi * wx^2;
end

function residuals =
ReprojectCorners(R,t,K,Corners_Dataset,dirLoc,standard_img,proj_img)

%Parameters of the image that needs to be projected
P = K * [R{proj_img}(:,1:2) t{proj_img}];
corners_proj_img = Corners_Dataset{proj_img};
corners_proj_img = [corners_proj_img ones(size(corners_proj_img,1),1)];

% Read the image file which needs to be back projected
filename = strcat([dirLoc, '/Pic_' num2str(standard_img), '.jpg']);
Colorimg = imread(filename);
img = rgb2gray(Colorimg);
%Parameters of the fixed image
P_stand = K * [R{standard_img}(:,1:2) t{standard_img}];
% corners of fixed image
corners_fixed = Corners_Dataset{standard_img};
%Projected corners
worldCoordImg = pinv(P) * corners_proj_img';
Corners_Projected = (P_stand * worldCoordImg)';

figure
imshow(Colorimg)
%Loop trhough all the corners
for i = 1:80
    %HC to XY
    Corners_Projected(i,:) = Corners_Projected(i,:) / Corners_Projected(i,3);
    hold on
    %plot the circle of size 16 at corners in green and red fro projected

plot(uint64(corners_fixed(i,1)),uint64(corners_fixed(i,2)),'g.','MarkerSize',
16);

plot(uint64(Corners_Projected(i,1)),uint64(Corners_Projected(i,2)),'r.','Mark
erSize',16);
end
hold off
%Compute vital statistics to evalute the performance of computed parmeters
diff = corners_fixed-Corners_Projected(:,1:2);
%Calculating moments of the error
mean_err = mean(abs(diff(:)));
var_err = var(abs(diff(:)));
residuals = [mean_err, var_err];
end
```