ECE661: Computer Vision (Fall 2014)

Shaobo Fang: s-fang@purdue

November 18, 2014

Contents

1 Overview

In this assignment Zhang's camera calibration algorithm will be implemented. In order to achieve optimized results, non-linear optimization will be implemented after least square for estimating the intrinsic and extrinsic parameters.

In order to have a good estimation of good homographies of each view with regarding to the world coordinates, we will establish 80 corners' correspondences on the calibration pattern to the world coordinates. Note the corners established does not need to be 100 % correct for the following reason:

1. We have larger amount of data set, for the testing data sets we have 40 different views and that is a total of $80 \times 40 = 3200$ corners.

2. We will implement non-linear optimization `Levenberg Marquadt` algorithm.

The corner correspondences will be established using Canny Edge operator to extract the edge first and then using Hough Transform to extract the lines. Then the vertical and horizontal lines will be grouped and corners will be the interception points.

Thus, as the corner is obtained using interception points, the corners estimated will have sub-pixel accuracy.

A unique label will then be assigned to each of the corner following the left to right, bottom to top order. Intrinsic and extrinsic parameters will then be estimated based on the homography obtained using corner correspondences.

In order to evaluate the performance, we will project the points back to the fix images (self picked). If the re-projection is close to the original corners detected, then it means the intrinsic and extrinsic parameters are accurate.

Finally, the euclidean distances and the radial distortion will also be estimated to formulate numerical error analysis.

2 Canny Edge Detection, Hough Transform and Corner Labelling

2.1 Canny Edge Detection

In the assignment the Canny Edge Detection was fulfilled using the Matlab function `edge(input, 'canny')`. In general, a Canny edge detector was implemented following below procedures `Reference:[I]`:

1. Gaussian Filter to remove noise. As Canny edge operator is very sensitive to noise first of all the gray-scale image need to be filtered to remove noise. Note that the sum of Gaussian kernel should be equal to 1.

2. Find the intensity of gradient using Sobel operator (or similar gradient operator). After d_x and d_y is obtained, we need to estimate the strength and the direction of the gradient at each of the pixel location:

$$Gradient\ Strength = \sqrt{d_x^2 + d_y^2}$$

$$Gradient\ Direction = \arctan(\frac{d_y}{d_x})$$

   Group the gradient direction into 4 (0, 90, 180 or 270 degrees) or 8 angles (0, 45, 90, 135, 180, 235, 270 or 315 degrees).

3. Non-maxima suppression in order to extract the edges and suppress local non-maxima gradient strength.

4. Finally, edge detection using two threshold method.

$$If\ gradient\ strength \geq T_{high} \rightarrow edge\ established$$

$$If\ gradient\ strength \leq T_{low} \rightarrow edge\ rejected$$

Otherwise, pixel will be accepted as edge iff it is connected to accepted edge.

`For the Canny edge extracted please refer to the results section.`

`[I]: OpenCV Canny Edge Detector: http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html`

2.2 Hough Transform and Corner Labelling

The Hough Line transformation was performed using matlab function: `houglines()`. Note that the input to the function is the Canny Edge extracted from the previous step. Accurate Canny Edges will have a better chance of yielding the correct hough lines.

From the calibration pattern, it can be easily seen that a total of 18 straight lines is required, no more, no less to calculate all the 80 corners on the calibration pattern. There will be 8 vertical lines and 10 horizontal lines.

For each of the lines estimated using hough transform, it can be represented as either of the form below:

$$y = ax + b,\ if\ a \neq \infty$$

$$x = b,\ if\ a = \infty$$

Now, as we pre-defined that there will be 8 vertical lines and 10 horizontal lines, the 10 hough lines with smaller $|a|$ will be grouped as horizontal lines. We simply need to determine sort the 10 horizontal lines based on b value so we can arrange the lines from the bottom to the top. Find all 8 interception points for each of the horizontal lines with regarding to the vertical lines and again, sort those points from left to right order we will have the correct label.

Note that this implementation will assign a unique number to a corner all the time. However as the calibration pattern is bottom-top symmetric we do not care the bottom-top flip. Excellent corners detection could be seen in the result session, and the corners detected is actually within **sub-pixel accuracy**. Due to the high accuracy of the corner detection, the **LM Algorithm** could not improve our result significantly, (as the result is already very close to perfect).

3 Zhang's Camera Calibration

By Implementing the camera calibration algorithm , the intrinsic parameters and extrinsic parameters of a camera for a specific picture could be estimated.

Assume that a point X in the real world plane have the following coordinate:

$$X^{world} = [x, y, z, 1]^T$$

Then, the coordinates projected on the 2D image plane (on Camera sensors) will be:

$$X^{image} = H^{3\times4}X^{world},\ while\ X^{image} = [x', y', 1]^T$$

$$X^{image} = [h_1, h_2, h_3, h_4] * X^{world}$$

Decompose the Homography into intrinsic and extrinsic parameters:

$$X^{image} = K^{3\times3} * [R|\vec{t}] * X^{world}$$

Now, we consider the point in the real world plane is located at $z = 0$ plane, accordingly:

$$X^{image} = K^{3\times3} * [R|\vec{t}] * [x, y, 0, 1]^T$$

Now, we can easily see that Homography H become a 3 by 3 matrix.

$$H^{3\times3} = [h_1, h_2, h_4]$$

Or, simply:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}^{image} = [h_1, h_2, h_4] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}^{world}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}^{image} = K[r_1, r_2, r_4] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}^{world}$$

Now, we noticed the point in 3D is represented in 2D as it is assumed to be located on the $z = 0$ plane.

Now, from lecture notes we know that assuming the pattern is visually rich enough so the homography could be estimated based on feature points. Then, applying H to $\vec{I}$ and $\vec{J}$, we can formulate:

$$H\vec{I} = \vec{h}_1 + i\vec{h}_2$$

$$H\vec{J} = \vec{h}_1 - i\vec{h}_2$$

$$(H\vec{I})^T W (H\vec{I}) = 0$$

$$(H\vec{J})^T W (H\vec{J}) = 0$$

$$(\vec{h}_1 + i\vec{h}_2)^T W(\vec{h}_1 + i\vec{h}_2) = 0$$
$$(\vec{h}_1 - i\vec{h}_2)^T W(\vec{h}_1 - i\vec{h}_2) = 0$$

Then, easily:

$$\vec{h}_1^T w\vec{h}_1 - \vec{h}_2^T w\vec{h}_2 = 0$$
$$\vec{h}_1^T w\vec{h}_2 = 0$$

In our implementation, we assume the image coordinates are the pixel locations of the corners detected in previous step and the world coordinates are as followed: $[0,0], [24.5,0], [49,0], ...[0,24.5], [0,49], ...[196,245]$ (as the distance measured between successive corners on the calibration pattern is 24.5mm).

Now, let us focus on the matrix:

$$W = \begin{bmatrix} w_{11}, w_{12}, w_{13} \\ w_{21}, w_{22}, w_{23} \\ w_{31}, w_{32}, w_{33} \end{bmatrix}$$

As an image conic in its homogeneous representation is always a 3×3 symmetric matrix, we have only six unknowns in the above matrix W.

$$Define: \vec{b} = \begin{bmatrix} w_{11} \\ w_{12} \\ w_{22} \\ w_{13} \\ w_{23} \\ w_{33} \end{bmatrix}$$

The problem mow become how to estimate the vector $\vec{b}$.

As

$$\vec{b}_1^T W\vec{b}_2 = 0$$

We can get

$$(h_{11}, h_{12}, h_{13}) \begin{bmatrix} w_{11}, w_{12}, w_{13} \\ w_{21}, w_{22}, w_{23} \\ w_{31}, w_{32}, w_{33} \end{bmatrix} \begin{pmatrix} h_{21} \\ h_{22} \\ h_{23} \end{pmatrix} = 0$$

Assume:

$$\vec{v}_{12}^T \vec{b} = 0$$

Thus,

$$\vec{v}_{ij} = \begin{pmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{pmatrix}$$

Now, in the same manner for $\vec{h}_1^T w \vec{h}_1 - \vec{h}_2^T w \vec{h}_2 = 0$ it can be expressed as:

$$(\vec{v}_{11} - \vec{v}_{22})^T \vec{b} = 0$$

As we only have 6 unknowns, we only need three camera views. However, in order to improve accuracy we will use significantly more views (test data set: 40 views, my date set: 24 views) and have the parameters estimated using least squared method.

Intrinsic Parameters As $W = K^{-T}K^{-1}$ (where K is the intrinsic parameters), we have:

$$W = \begin{bmatrix} w_{11}, w_{12}, w_{13} \\ w_{21}, w_{22}, w_{23} \\ w_{31}, w_{32}, w_{33} \end{bmatrix} = \begin{bmatrix} \frac{1}{\alpha_x^2} & \frac{-s}{\alpha_x^2 \alpha_y} & \frac{y_0 s - x_0 \alpha_y}{\alpha_x^2 \alpha_y} \\ \frac{-s}{\alpha_x^2 \alpha_y} & \frac{s^2}{\alpha_x^2 \alpha_y^2} + \frac{1}{\alpha_y^2} & \frac{-s(y_0 s - x_0 \alpha_y)}{\alpha_x^2 \alpha_y} - \frac{y_0}{\alpha_y^2} \\ \frac{y_0 s - x_0 \alpha_y}{\alpha_x^2 \alpha_y} & \frac{-s(y_0 s - x_0 \alpha_y)}{\alpha_x^2 \alpha_y} - \frac{y_0}{\alpha_y^2} & \frac{(y_0 s - x_0 \alpha_y)^2}{\alpha_x^2 \alpha_y} + \frac{y_0}{\alpha_y^2} + 1 \end{bmatrix}$$

Then, the intrinsic parameters are defined as:

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

In the intrinsic parameters, s represent the skew coefficient between x and y axis, α_x and α_y represent the focal lengths with regarding to the pixels and x_0, y_0 represent the principal point with regarding to the coordinates on camera sensor.

$$\begin{aligned} x_0 &= \frac{w_{12}w_{13} - w_{11}w_{23}}{w_{11}w_{22} - w_{12}^2} \\ \lambda &= w_{33} - \frac{w_{13}^2 + x_0(w_{12}w_{13} - w_{11}w_{23})}{w_{11}} \\ \alpha_x &= \sqrt{\frac{\lambda}{w_{11}}} \\ \alpha_y &= \sqrt{\frac{\lambda w_{11}}{w_{11}w_{22} - w_{12}^2}} \\ s &= -\frac{w_{12}\alpha_x^2 \alpha_y}{\lambda} \\ y_0 &= \frac{s x_0}{\alpha_y} - \frac{w_{13}\alpha_x^2}{\lambda} \end{aligned}$$

Please note that as x and y axis are both subjectively defined by the user, sometimes the order of x and y axis could be flipped. However, for the same camera and same focal length, the numerical values for λ, $skew : s$, x_0, y_0 and α_x, α_y should stay the same all the time.

Extrinsic Parameters $R = [r_1, r_2, r_3]$ and $\vec{t}$ is denoted as the extrinsic parameters. While R is the rotation matrix and $\vec{t}$ is the pixel coordinate of the world origin.

Extrinsic parameters could be estimated as below:

$$\vec{r_1} = K^{-1}\vec{h}_1$$

$$\vec{r_2} = K^{-1}\vec{h}_2$$

$$\vec{r_3} = \vec{r_1} \times \vec{r_2}$$

$$\vec{t} = K^{-1}\vec{h}_3$$

4 Performance Evaluation, non-linear Refinement and Radial Distortion

Performance Evaluation
To estimate the performance of the intrinsic and extrinsic parameters, we need to pick a fixed image. The fixed images are picked of those with orthogonal view with respect to the calibration pattern plane.

After the fixed images have been picked, we will project the corners detected from other view within the same dataset to the fix-image of the dataset. There will be two steps:

1. Project the corners detected from the different views back to the world coordinates first (with $z = 0$).
2. Project the world coordinates to the fixed-images.

Please note that all homography used in this section will be based on intrinsic and extrinsic parameters. Thus if our parameters are accurate enough then our re-projected points should be close enough.

In order to quantify our error, we will estimate the euclidean distance of real corners detected on fixed-image with respect to those re-projected corners. Then the mean and variance of euclidean distance would be used as estimate to indicate the system's performance.

Refinement
Assume we have:

$$d^2_{geom} = \Sigma_i \Sigma_j ||\vec{x}_{ij} - \hat{\vec{x}}_{ij}||^2$$

Furthermore:

$$d^2_{geom} = \Sigma_i \Sigma_j ||\vec{x}_{ij} - K[R_i|t_i]\vec{x}_{m,j}||^2$$

Where:

1. $\vec{x}_{m,j}$: The j^{th} salient point on the calibration pattern
2. $\vec{x}_{i,j}$: The actual image point for $\vec{x}_{m,j}$ in the i^{th} position of the camera
3. $\hat{\vec{x}}_{ij}$: The project image point for $\vec{x}_{m,j}$ using P for i^{th} camera position
4. R_i: The rotation matrix for the i^{th} position of the camera
5. $\vec{t_i}$: The translation vector for the i^{th} position of the camera
6. K: The camera calibration matrix for the intrinsic parameters

The Levenber-Marquardt Algorithm will be utilized to optimize the parameters in our implementation.

Radial Distortion

Now, let $(\hat{x}_{rad}, \hat{y}_{rad})$ be the pixel coordinates that would be predicted including the radial distortion:

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

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

Where

$$r^2 = (\hat{x} - x_0)^2 + (\hat{y} - y_0)^2$$

In order to improve the accuracy, we can make k_1 and k_2 part of the optimization problem.

5 Results:

5.1 Provided Dataset With 40 Images

5.1.1 Canny Edges on: `Pic_11, Pic_12, Pic_13, Pic_14, Pic_15`

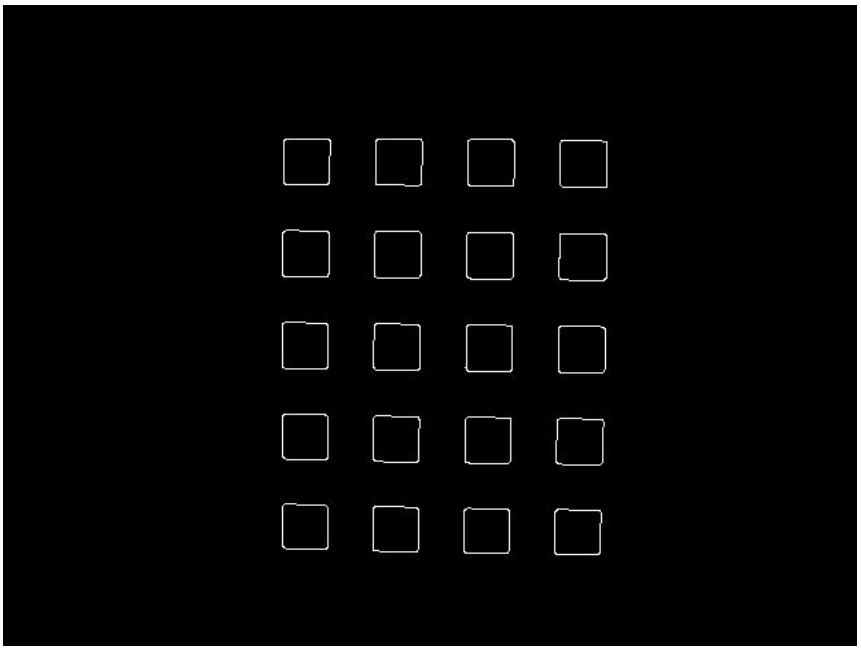

Fig 1. Canny Edges Extracted From Pic11.jpg

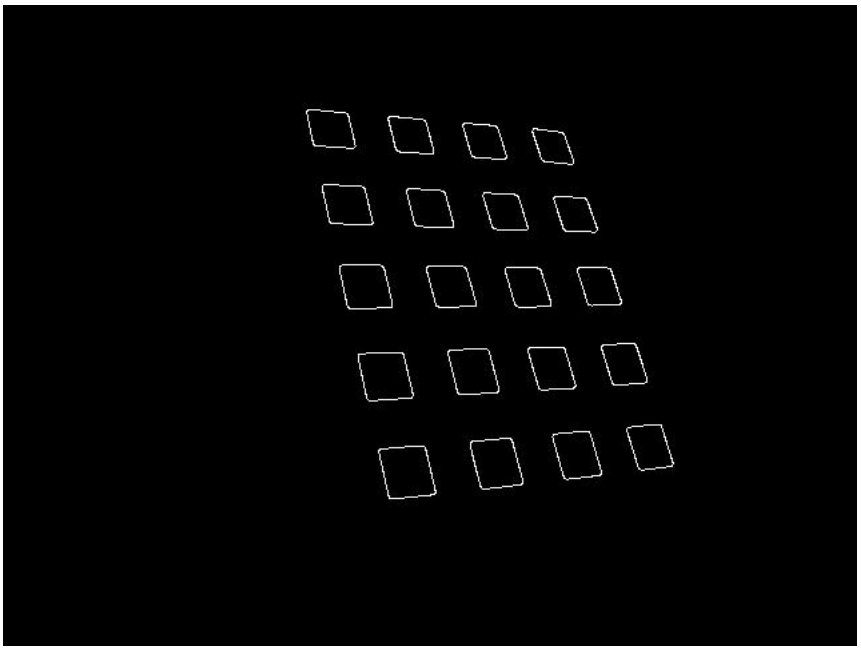

Fig 2. Canny Edges Extracted From Pic12.jpg

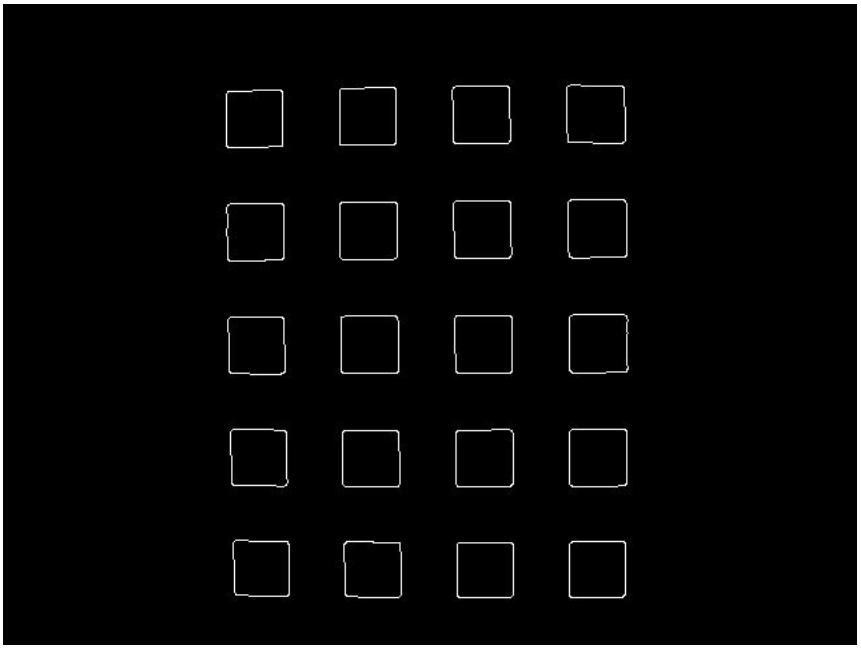

Fig 3. Canny Edges Extracted From Pic13.jpg

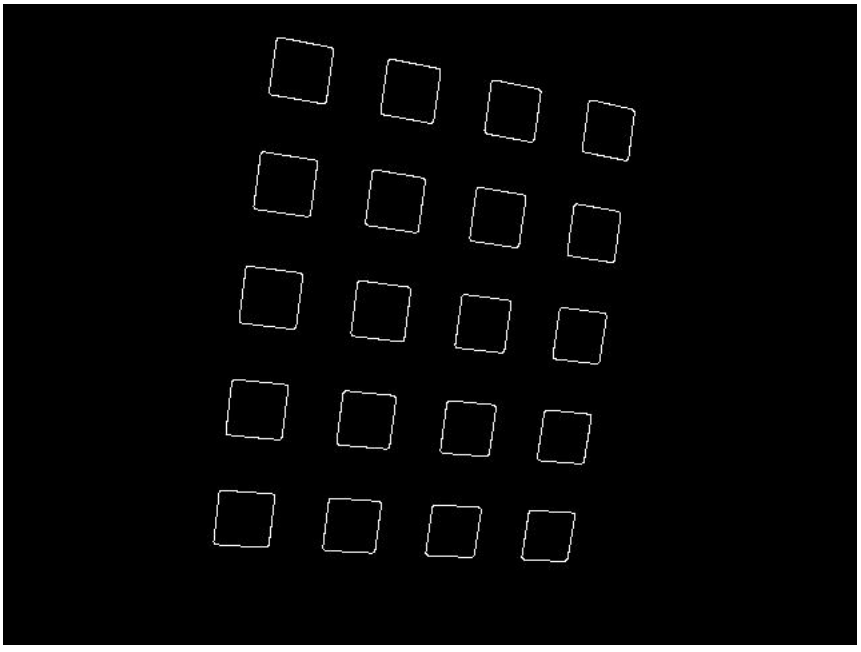

Fig 4. Canny Edges Extracted From Pic14.jpg

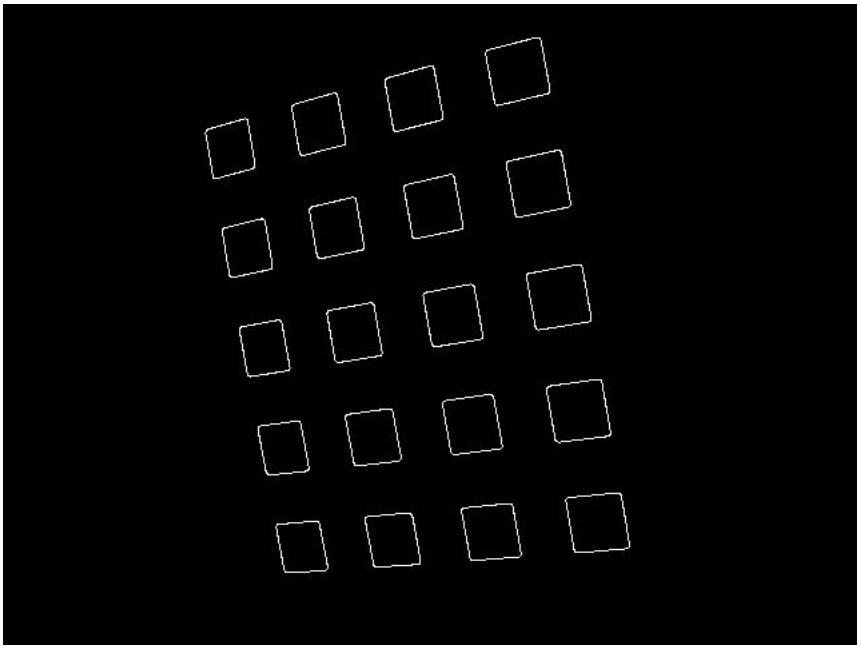

Fig 5. Canny Edges Extracted From Pic15.jpg

5.1.2 Hough Lines Fitting

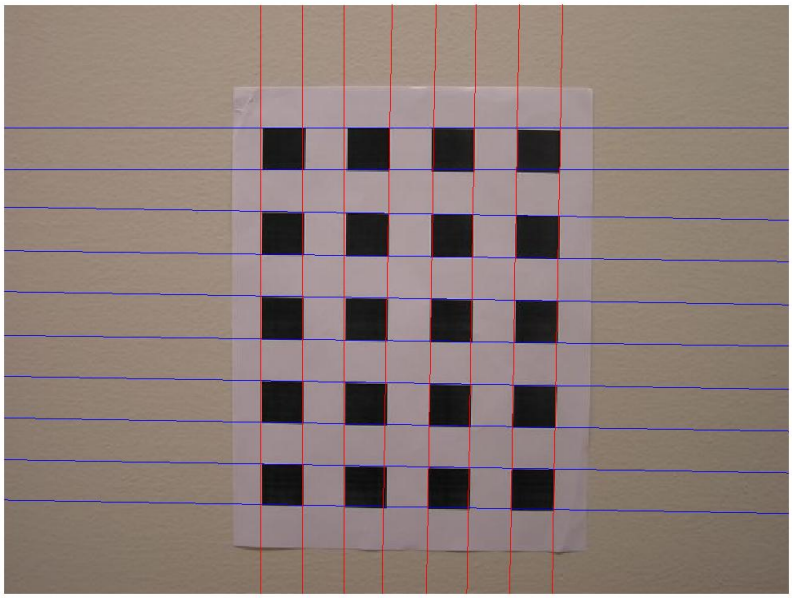

Fig 6. Hough Lines Fitting for Pic11.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

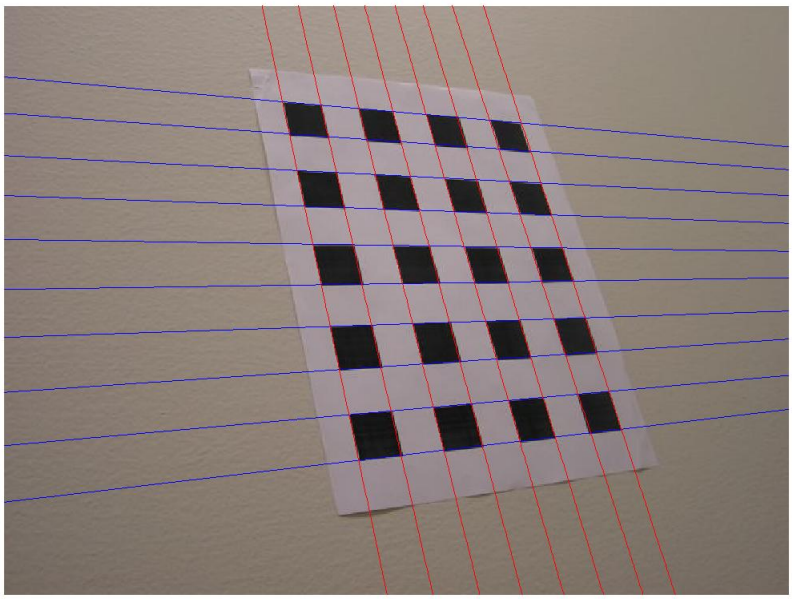

Fig 7. Hough Lines Fitting for Pic12.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

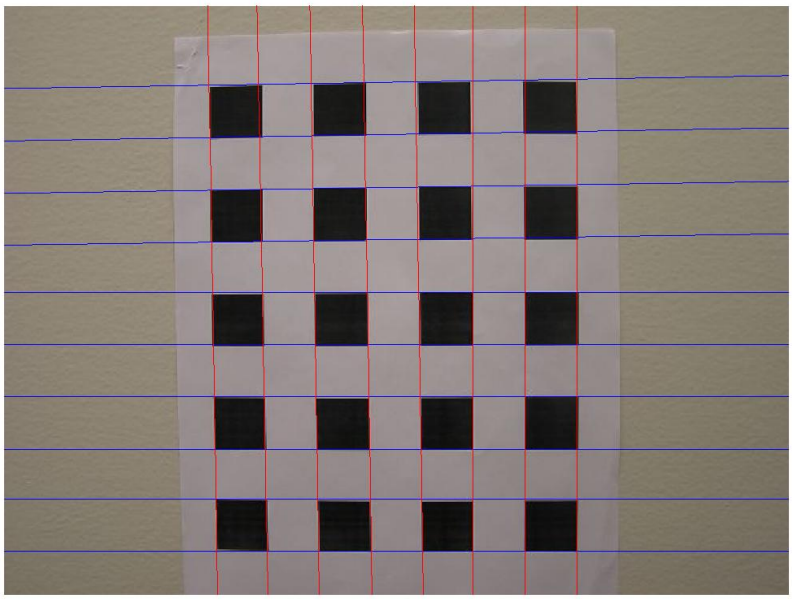

Fig 8. Hough Lines Fitting for Pic13.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

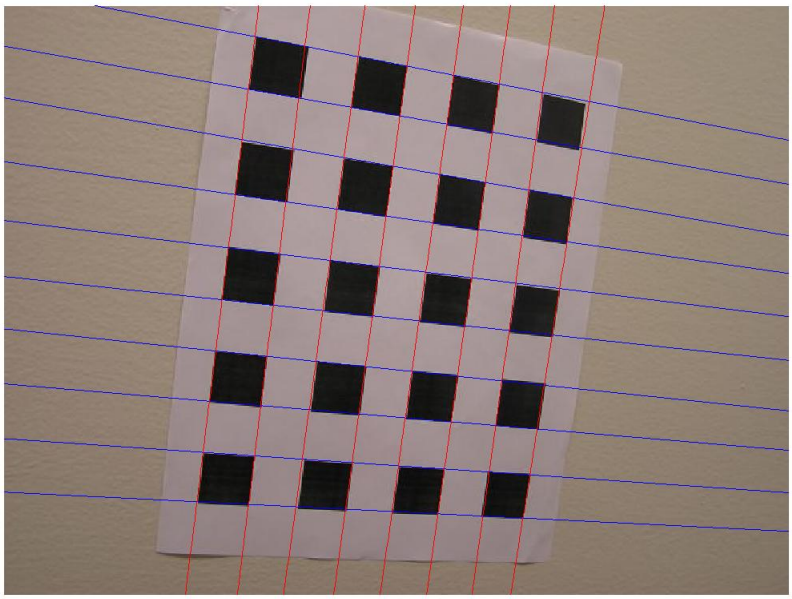

Fig 9. Hough Lines Fitting for Pic14.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

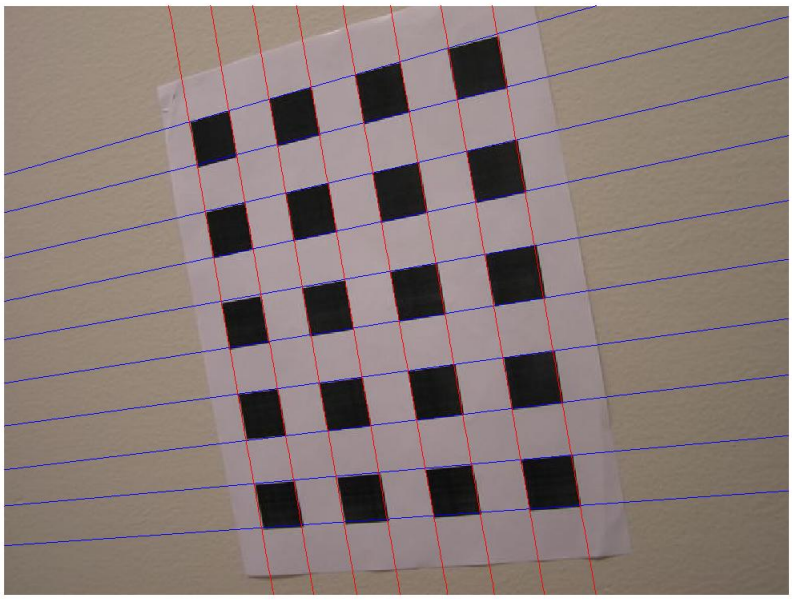

Fig 10. Hough Lines Fitting for Pic15.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

5.1.3 Corner Detection Based on Hough Lines

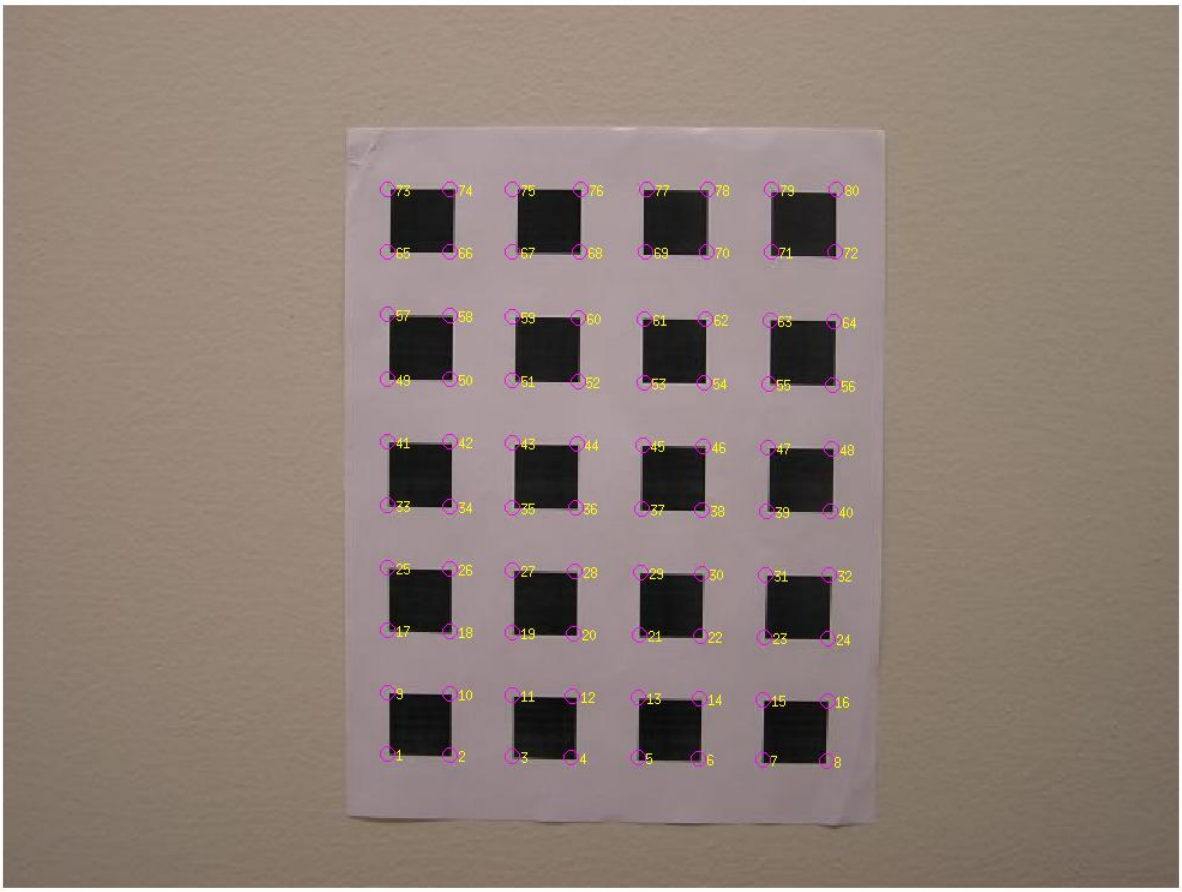

Fig 11. Corners detected based on Hough lines for Pic11.jpg

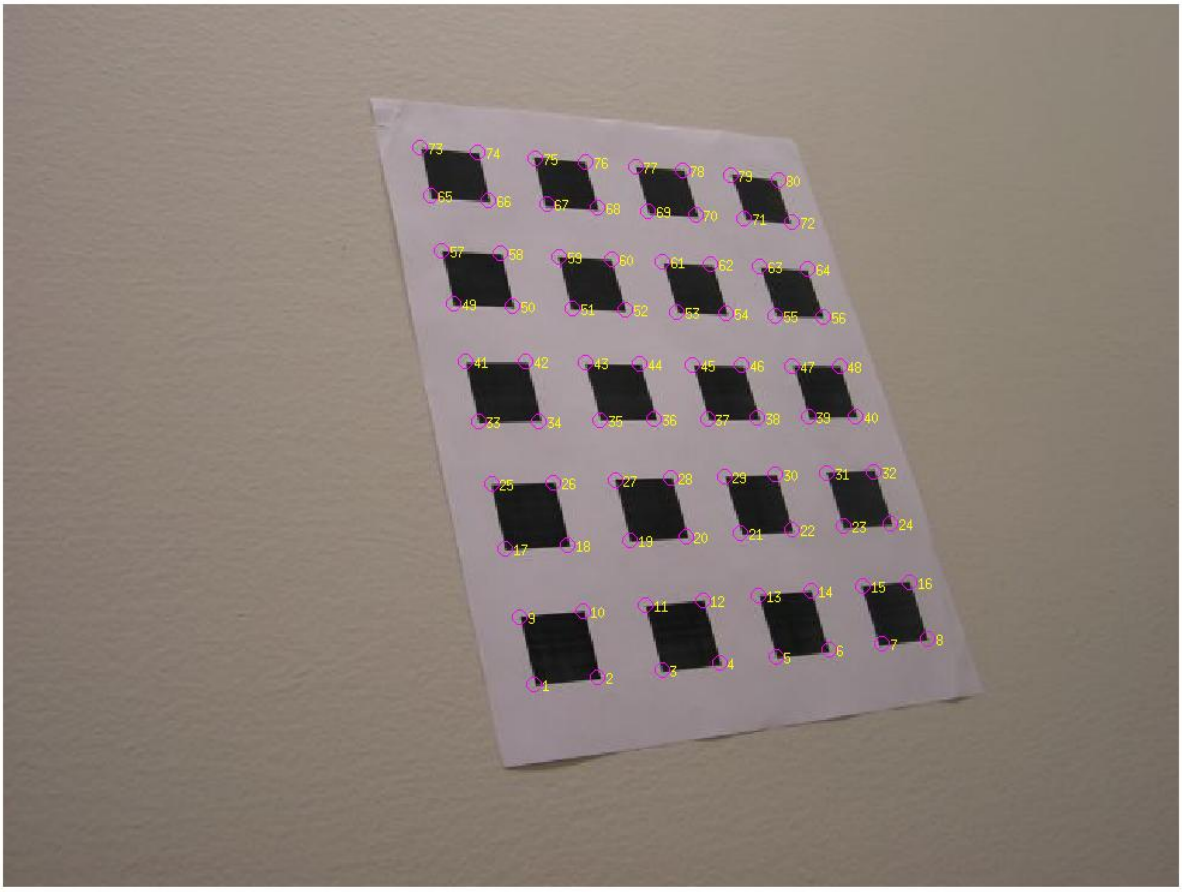

Fig 12. Corners detected based on Hough lines for Pic12.jpg

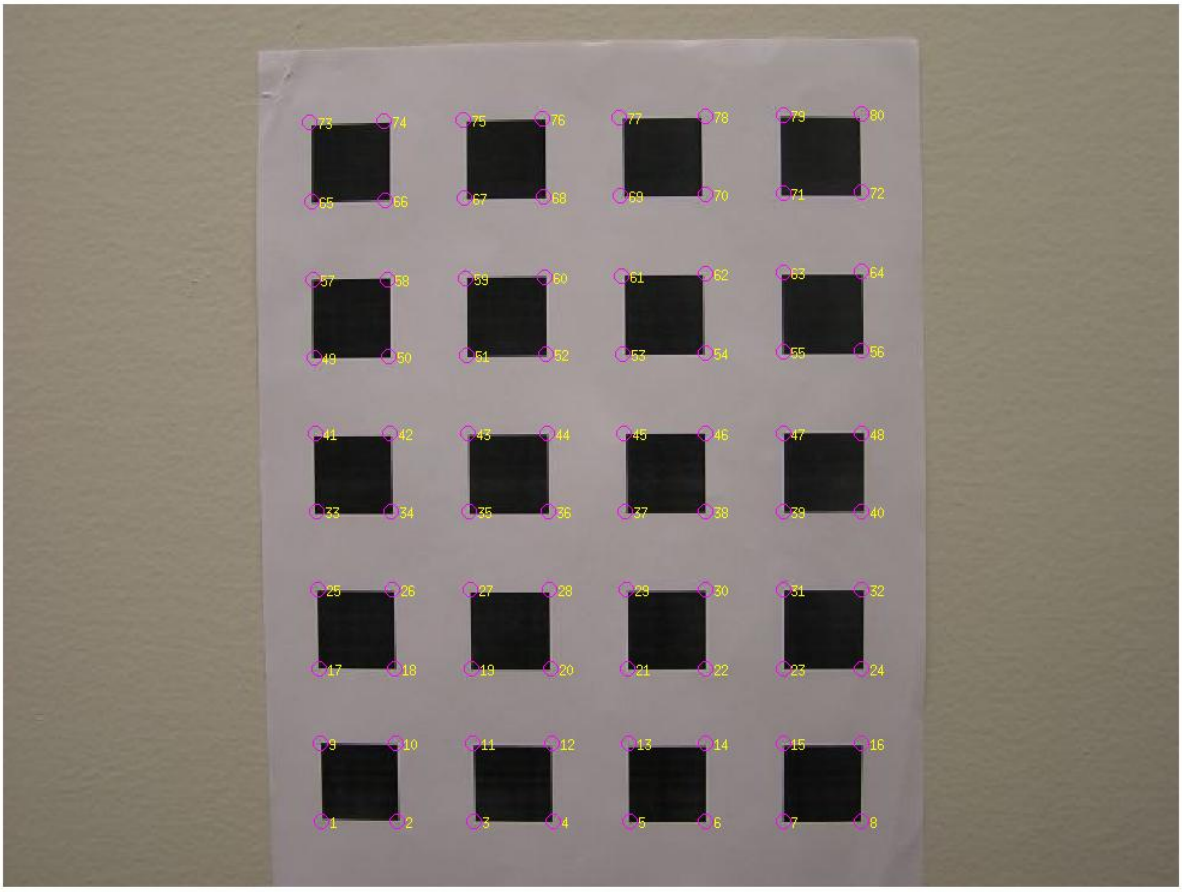

Fig 13. Corners detected based on Hough lines for Pic13.jpg

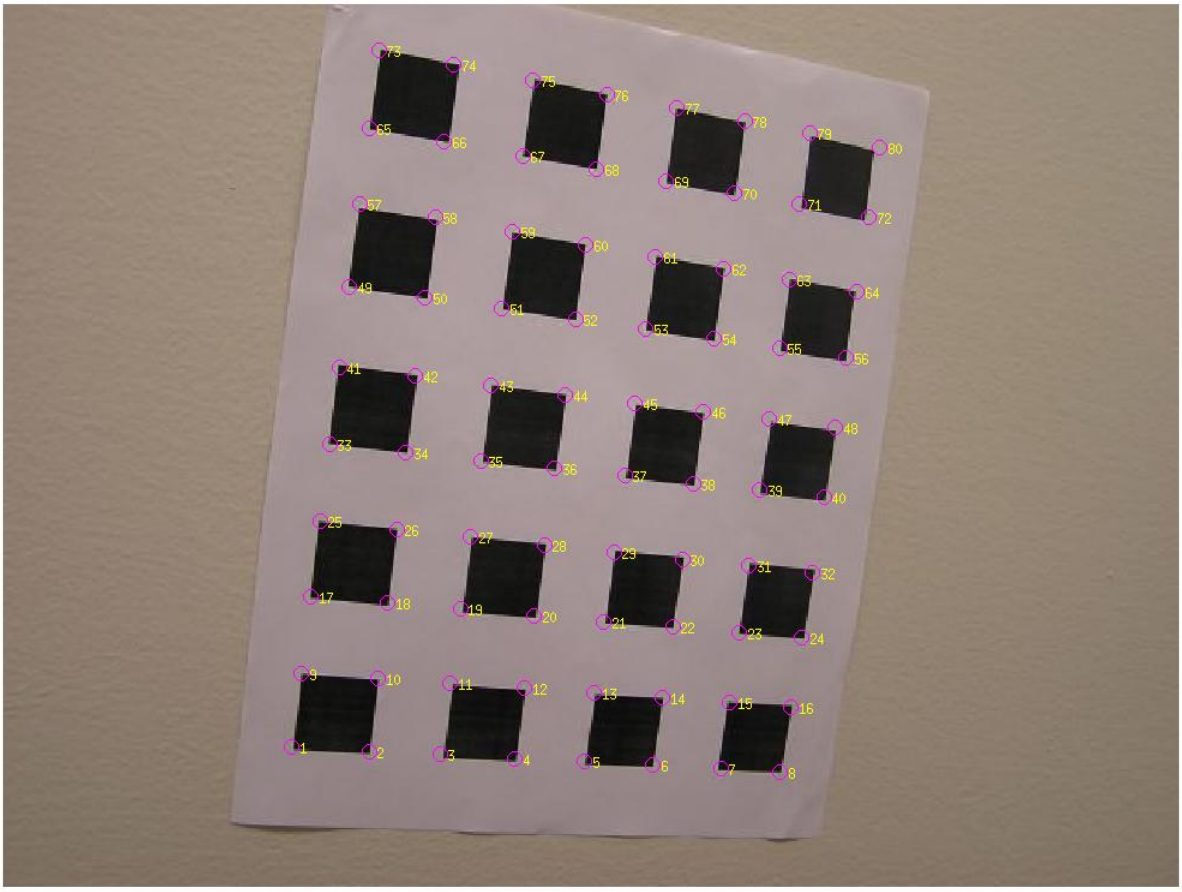

Fig 14. Corners detected based on Hough lines for Pic14.jpg

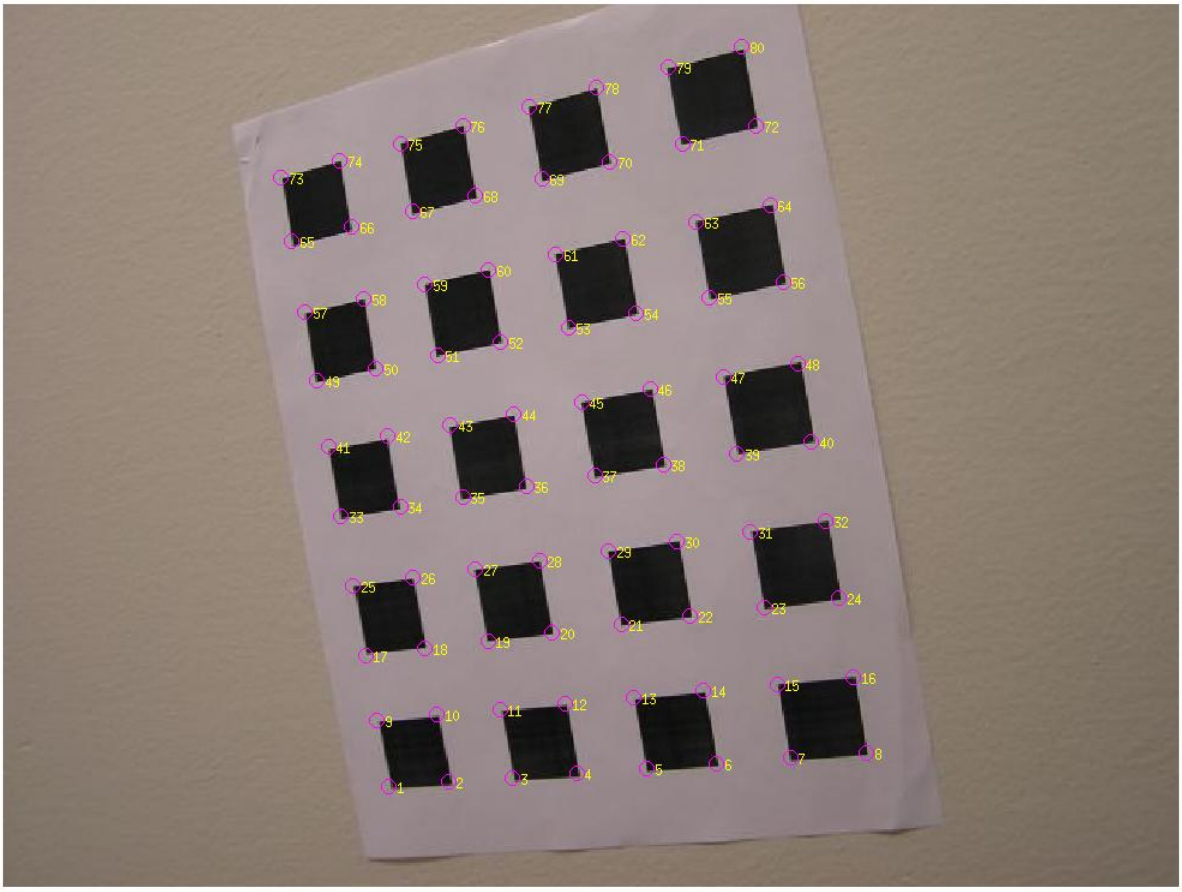

Fig 15. Corners detected based on Hough lines for Pic15.jpg

5.1.4 Re-Projected Corners with Regarding Fixed Image `Pic11.jpg`

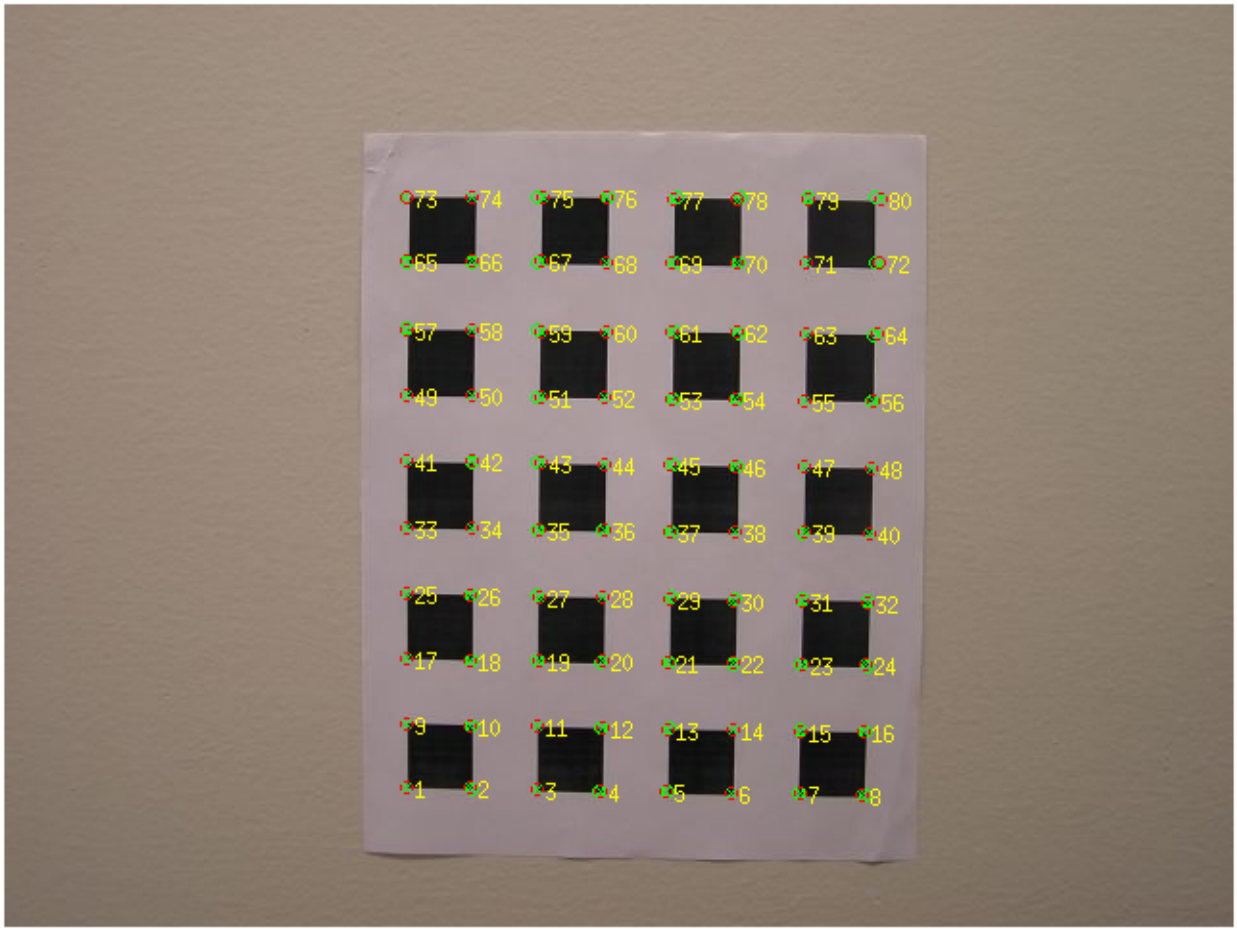

Fig 16. Re-project Corner on Pic12.jpg back to Pic11.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

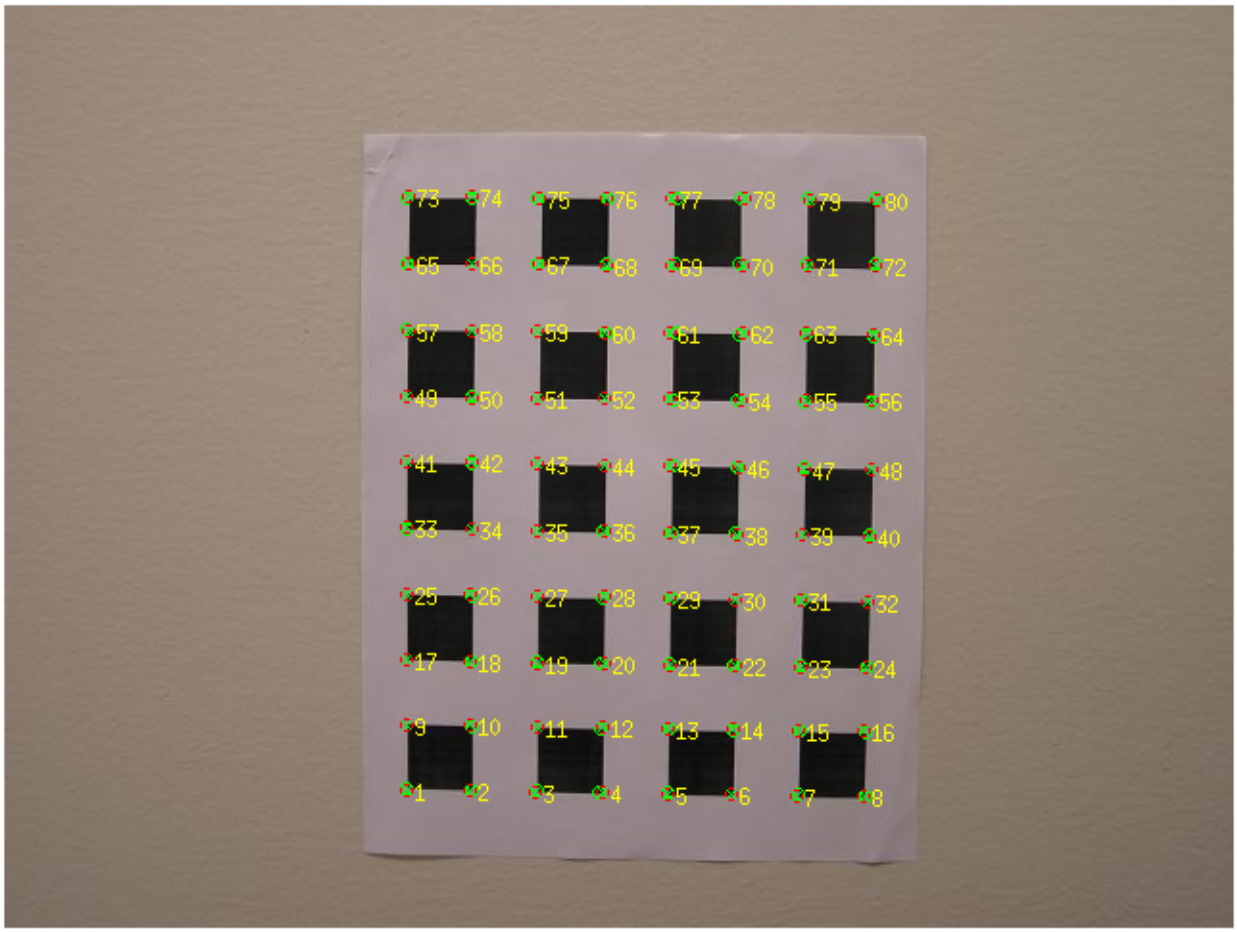

Fig 17. Re-project Corner on Pic13.jpg back to Pic11.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

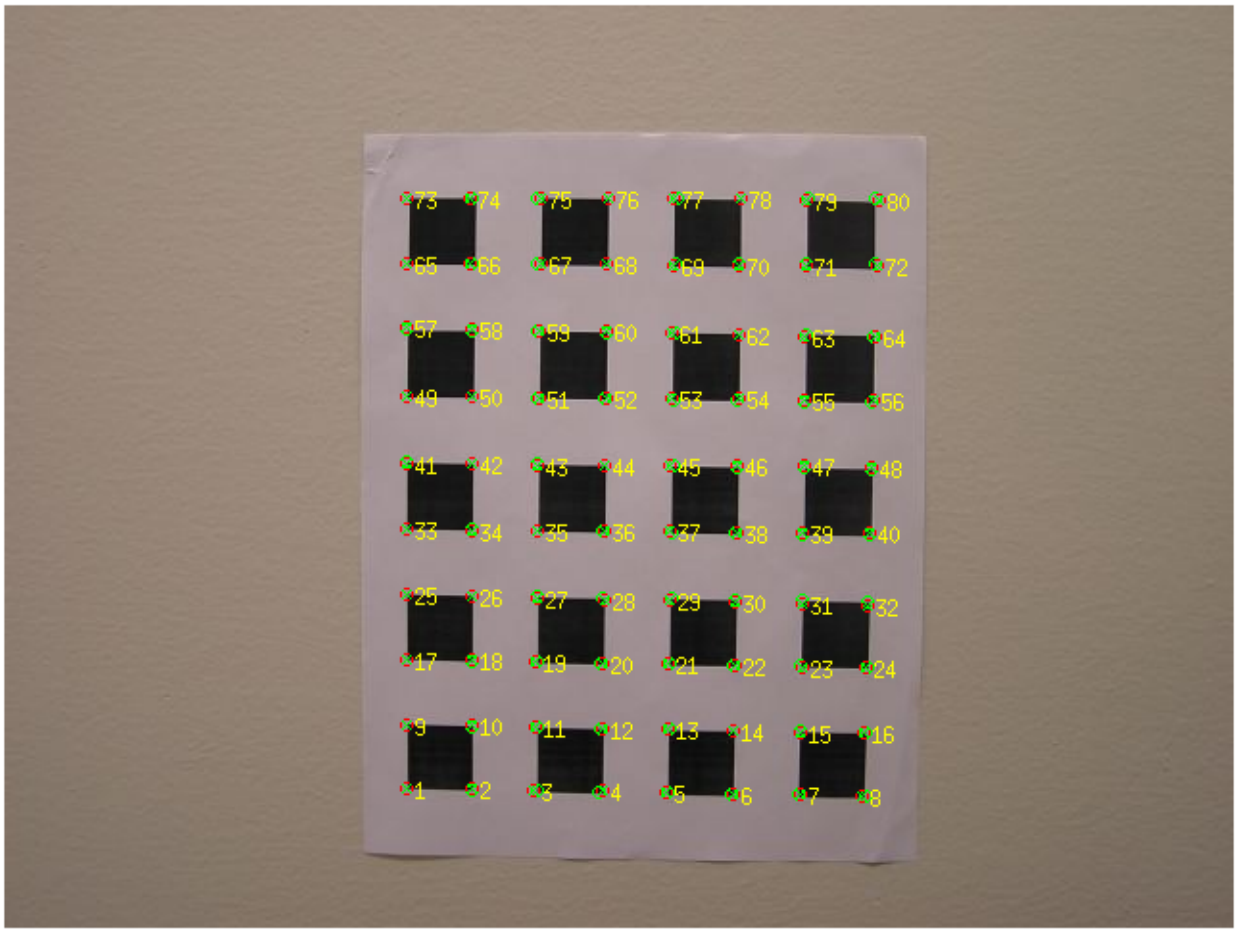

Fig 18. Re-project Corner on Pic14.jpg back to Pic11.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

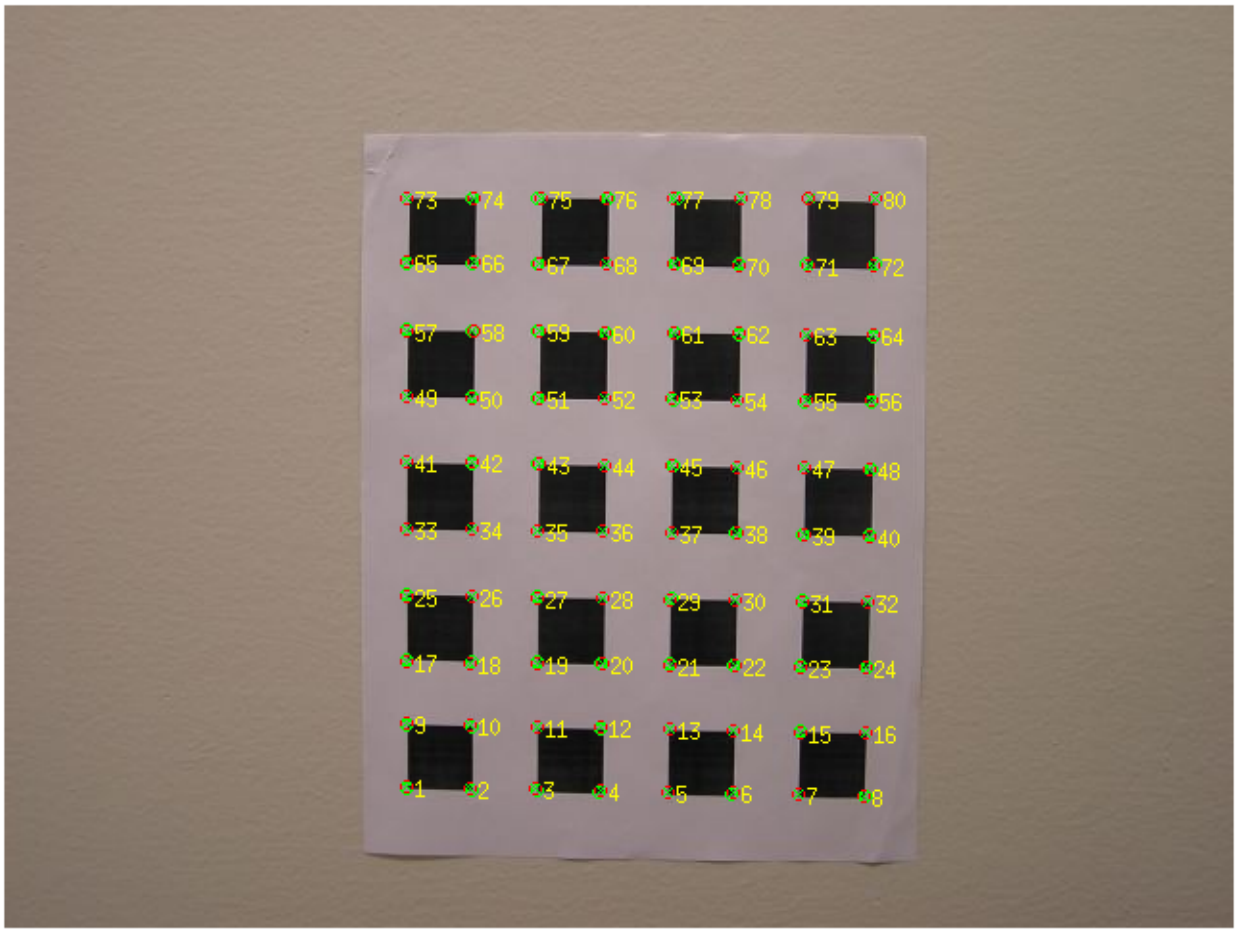

Fig 19. Re-project Corner on Pic15.jpg back to Pic11.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

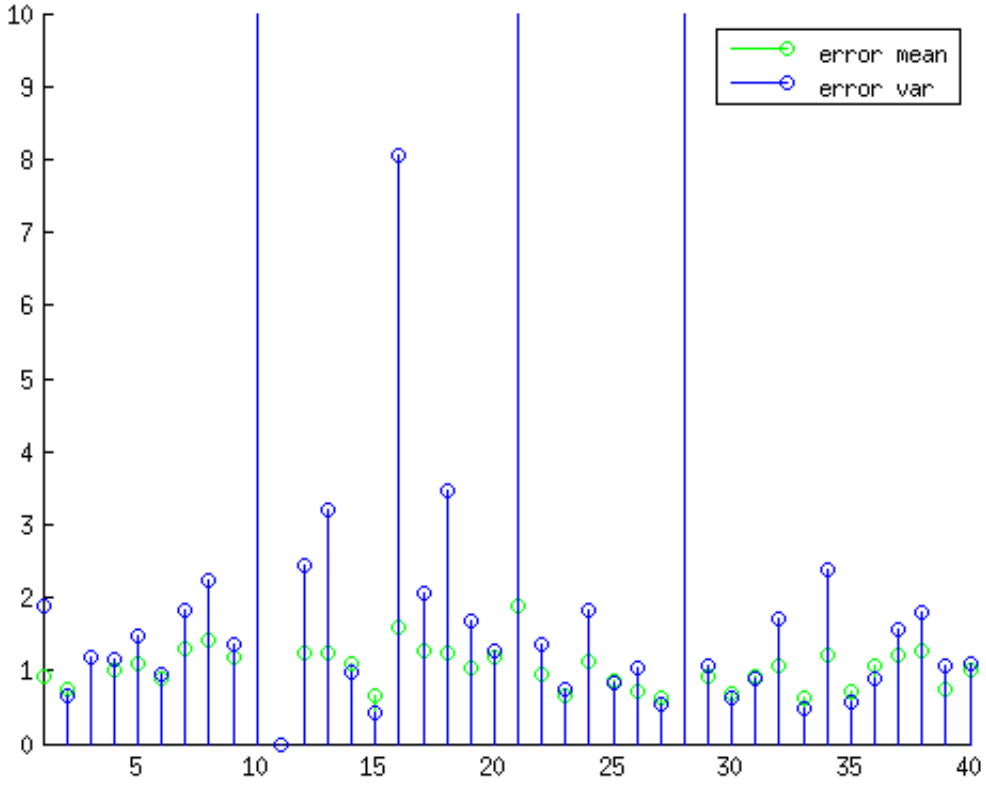

Fig 20. Quantitatively demonstrate the re-projection error.

DataSet1 Conclusion: From the plot above we can see the re-projection is usually very accurate with average error between 1-2 pixels. However, the re-projection for `Pic10.jpg` and `Pic28.jpg` is way off and would be further explained in the following session.

5.2 My Dataset With 24 Images

5.2.1 Canny Edges on: `Pic_20, Pic_21, Pic_22, Pic_23, Pic_24`

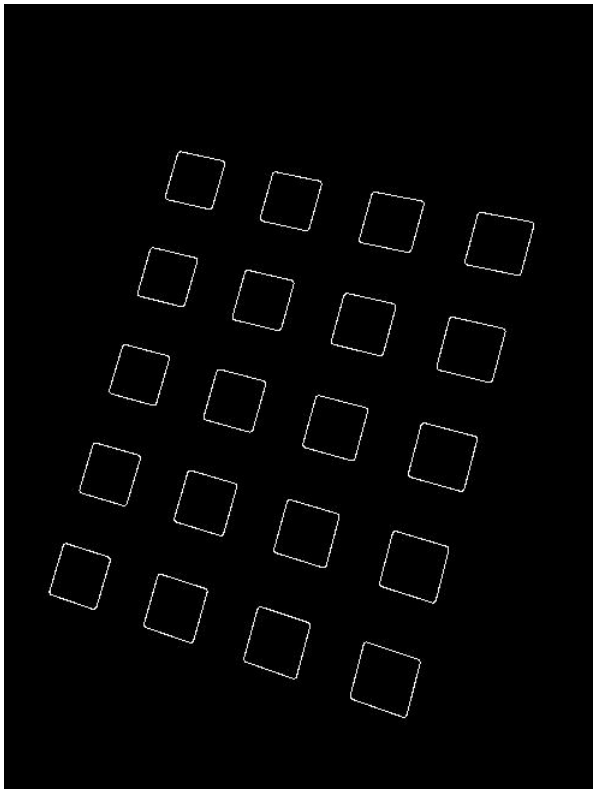

Fig 21. Canny Edges Extracted From Pic20.jpg

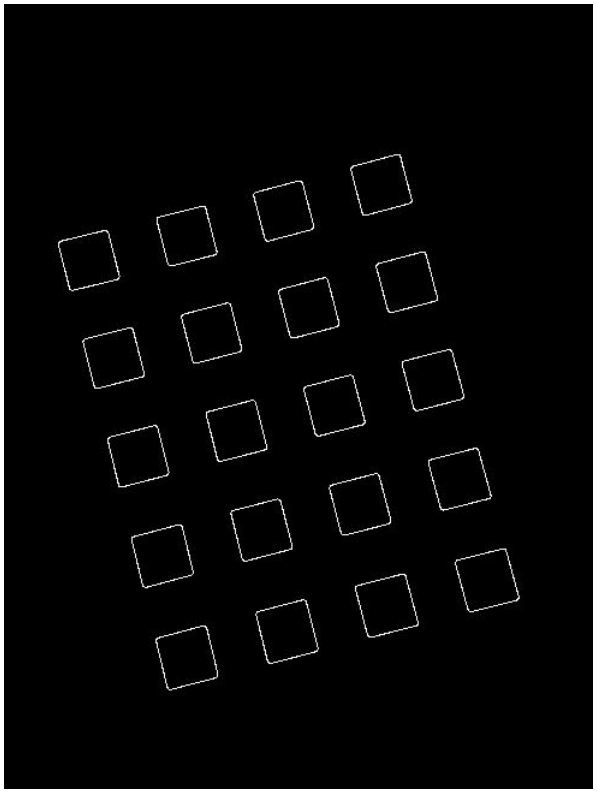

Fig 22. Canny Edges Extracted From Pic21.jpg

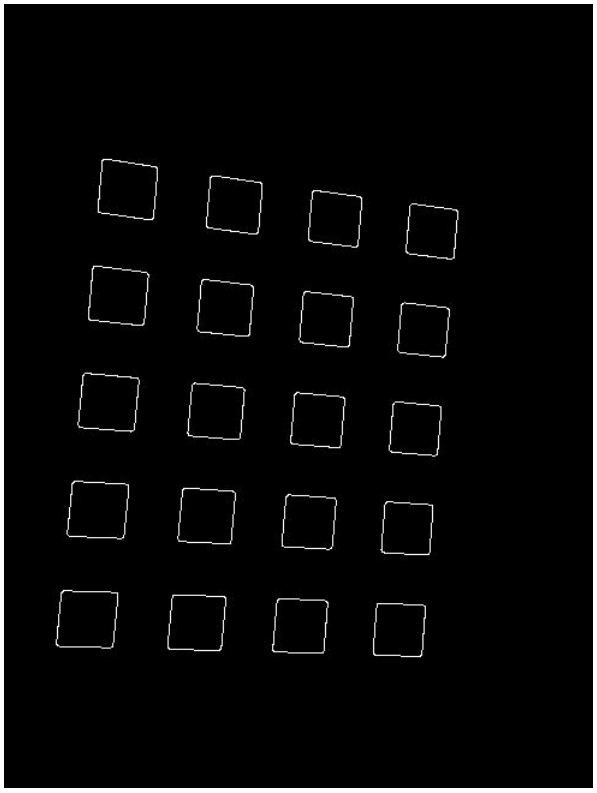

Fig 23. Canny Edges Extracted From Pic22.jpg

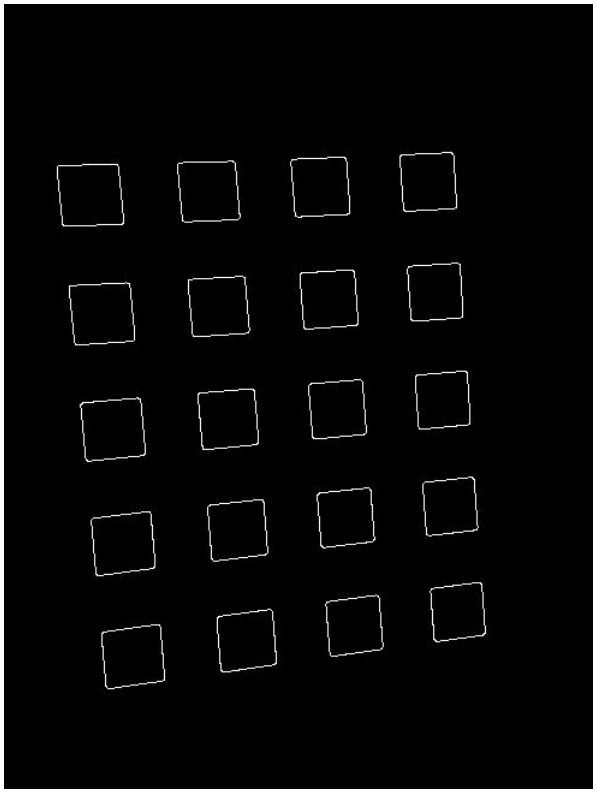

Fig 24. Canny Edges Extracted From Pic23.jpg

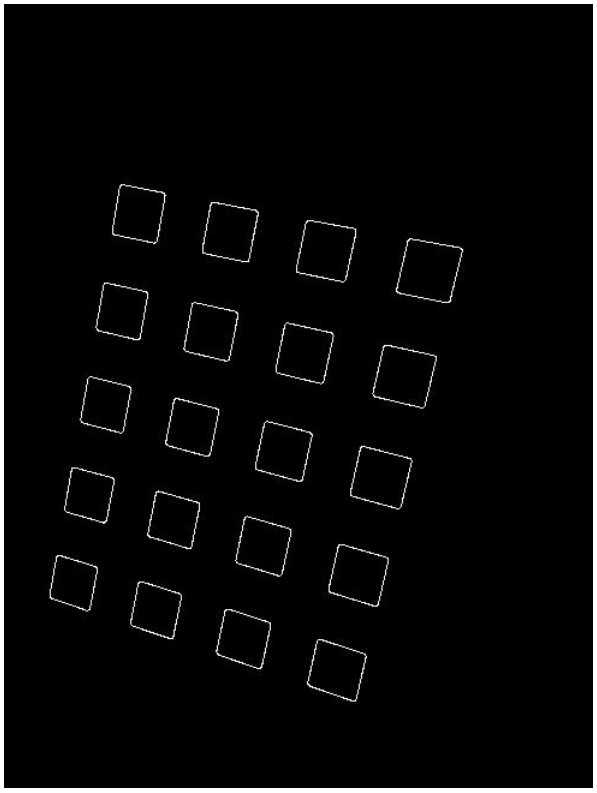

Fig 25. Canny Edges Extracted From Pic24.jpg

5.2.2 Hough Lines Fitting

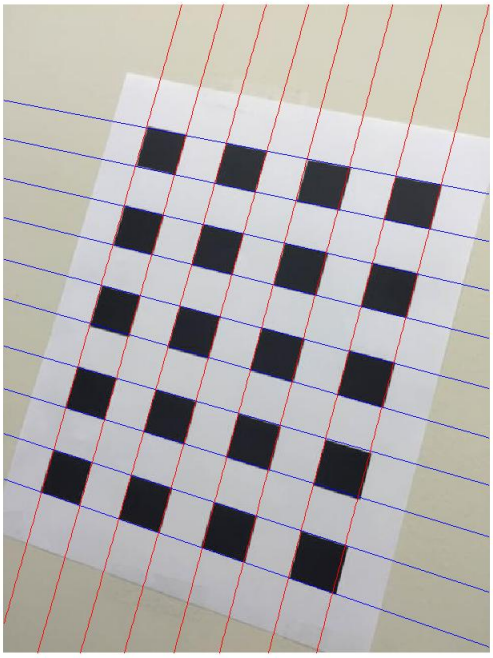

Fig 26. Hough Lines Fitting for Pic20.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

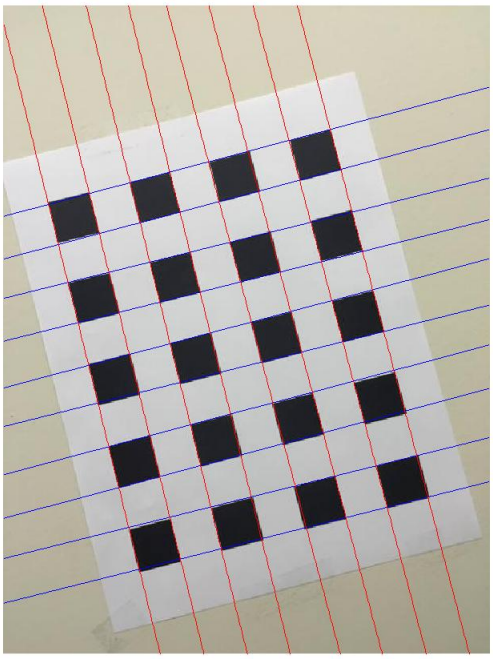

Fig 27. Hough Lines Fitting for Pic21.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

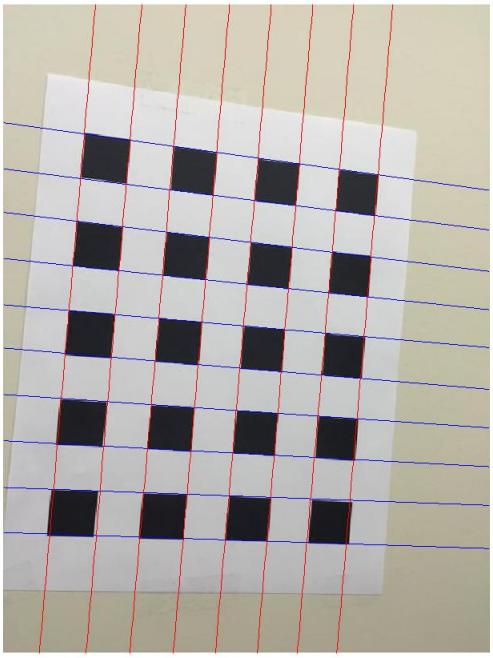

Fig 28. Hough Lines Fitting for Pic22.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

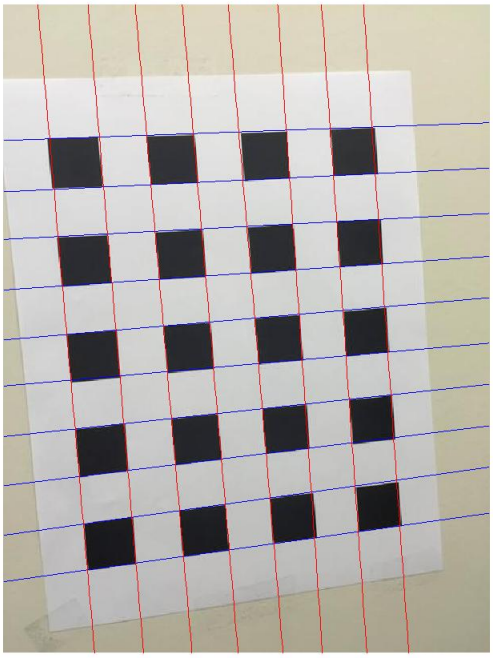

Fig 29. Hough Lines Fitting for Pic23.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

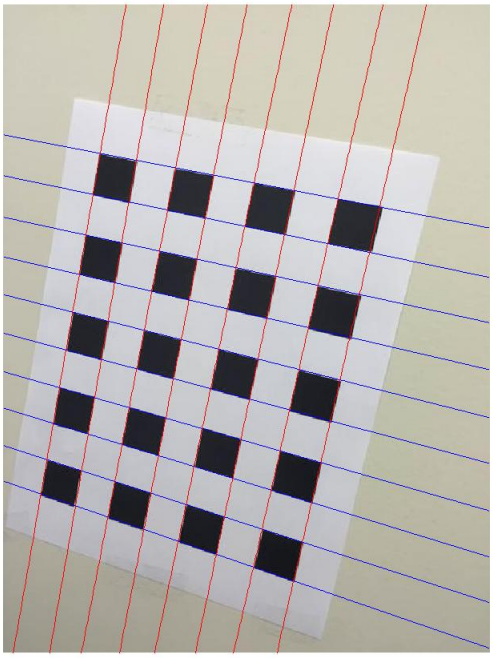

Fig 30. Hough Lines Fitting for Pic24.jpg, the red lines are considered to be vertical lines and blue lines are considered to be horizontal lines

5.2.3 Corner Detection Based on Hough Lines

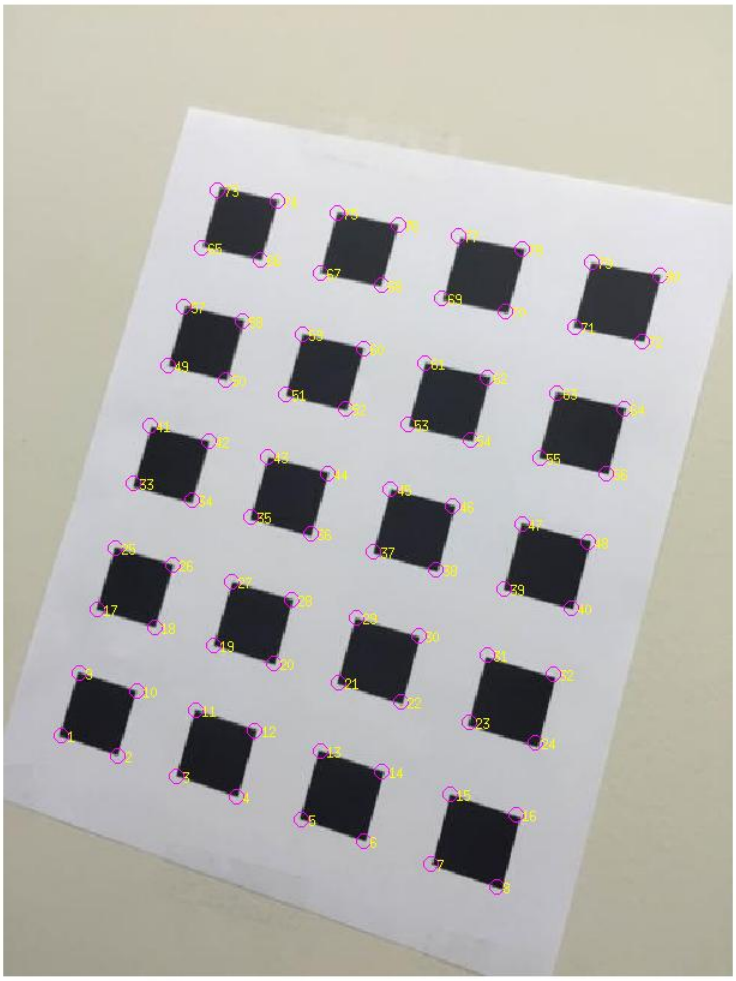

Fig 31. Corners detected based on Hough lines for Pic20.jpg

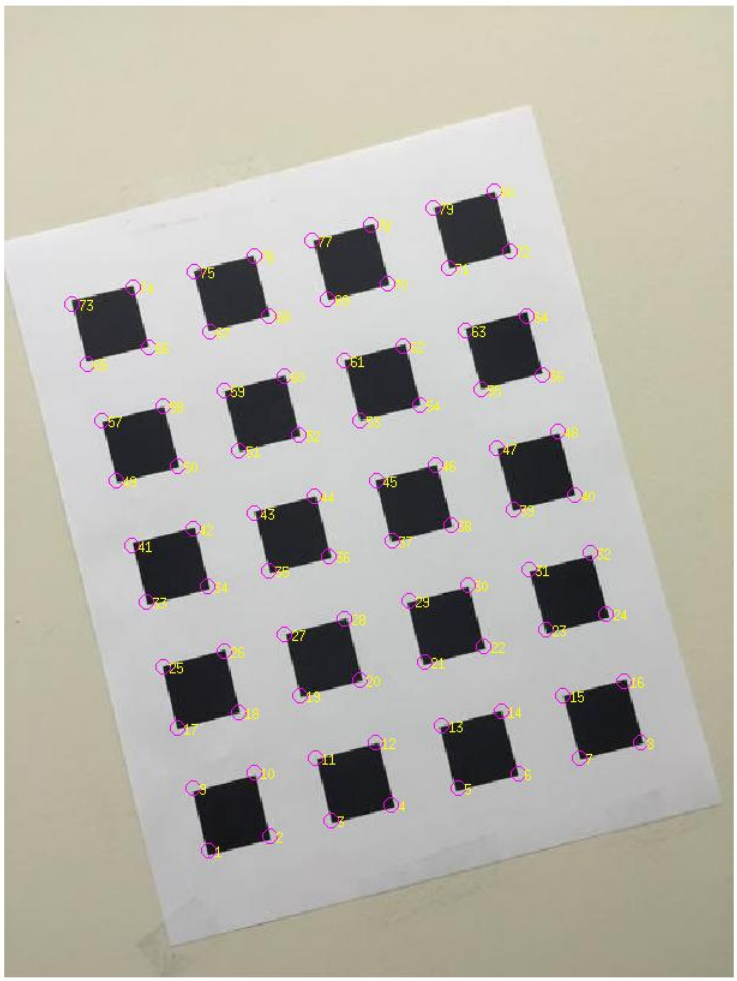

Fig 32. Corners detected based on Hough lines for Pic21.jpg

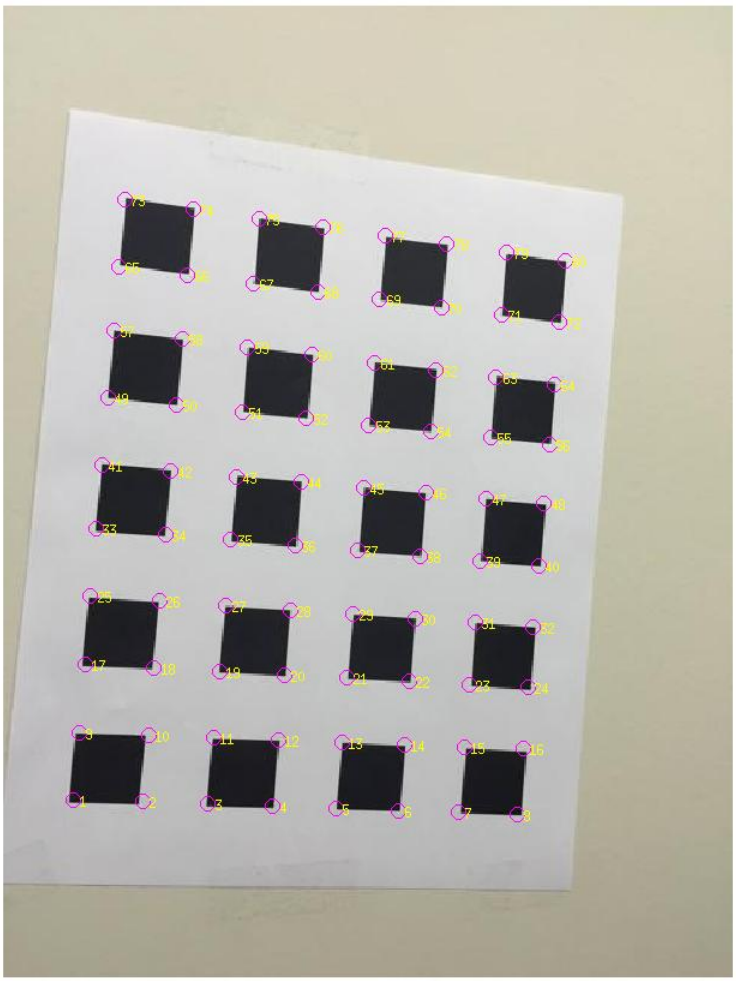

Fig 33. Corners detected based on Hough lines for Pic22.jpg

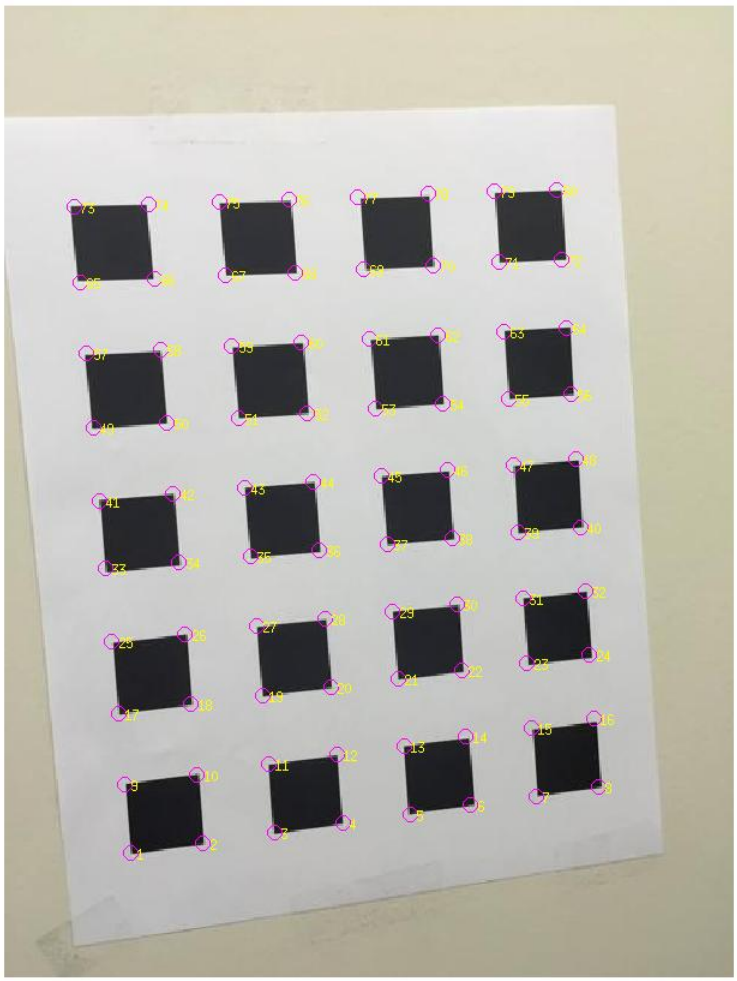

Fig 34. Corners detected based on Hough lines for Pic23.jpg

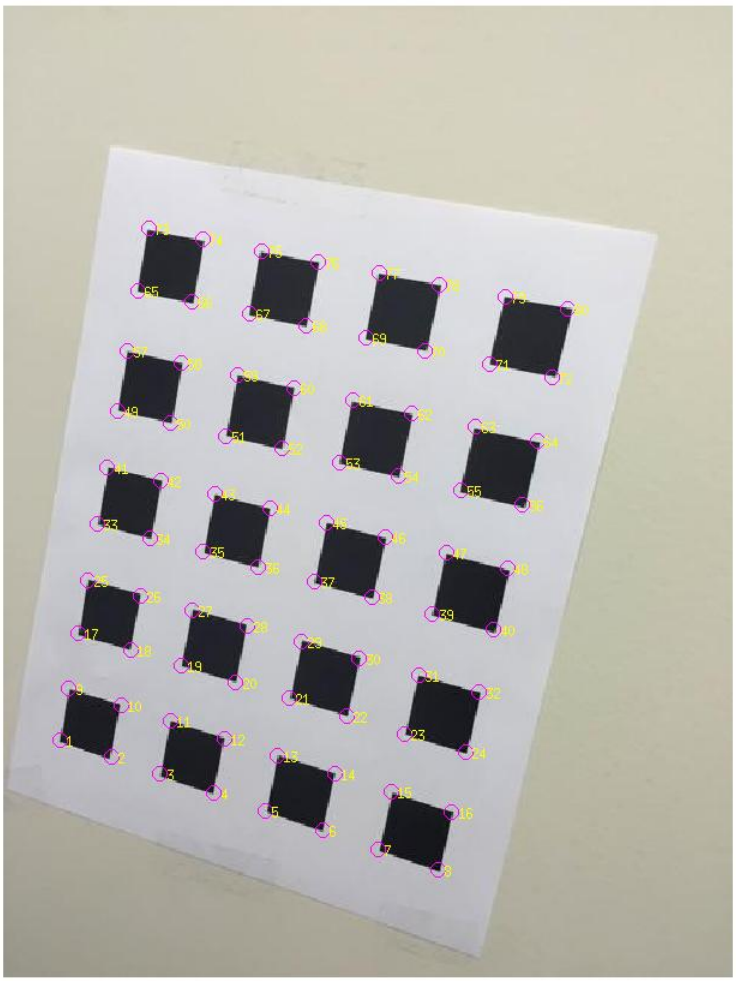

Fig 35. Corners detected based on Hough lines for Pic24.jpg

5.2.4 Re-Projected Corners with Regarding Fixed Image `Pic1.jpg`

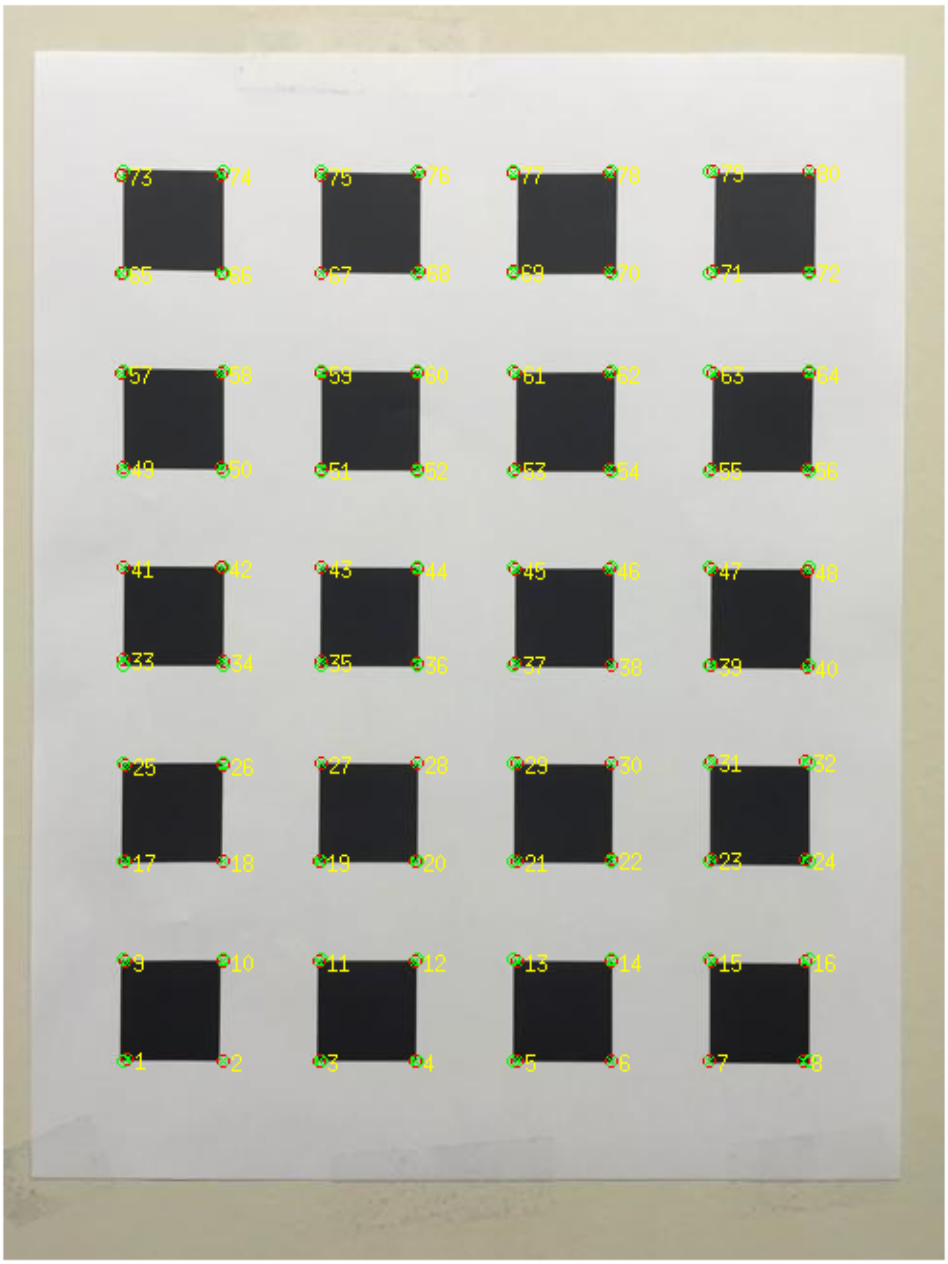

Fig 36. Re-project Corner on Pic20.jpg back to Pic1.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

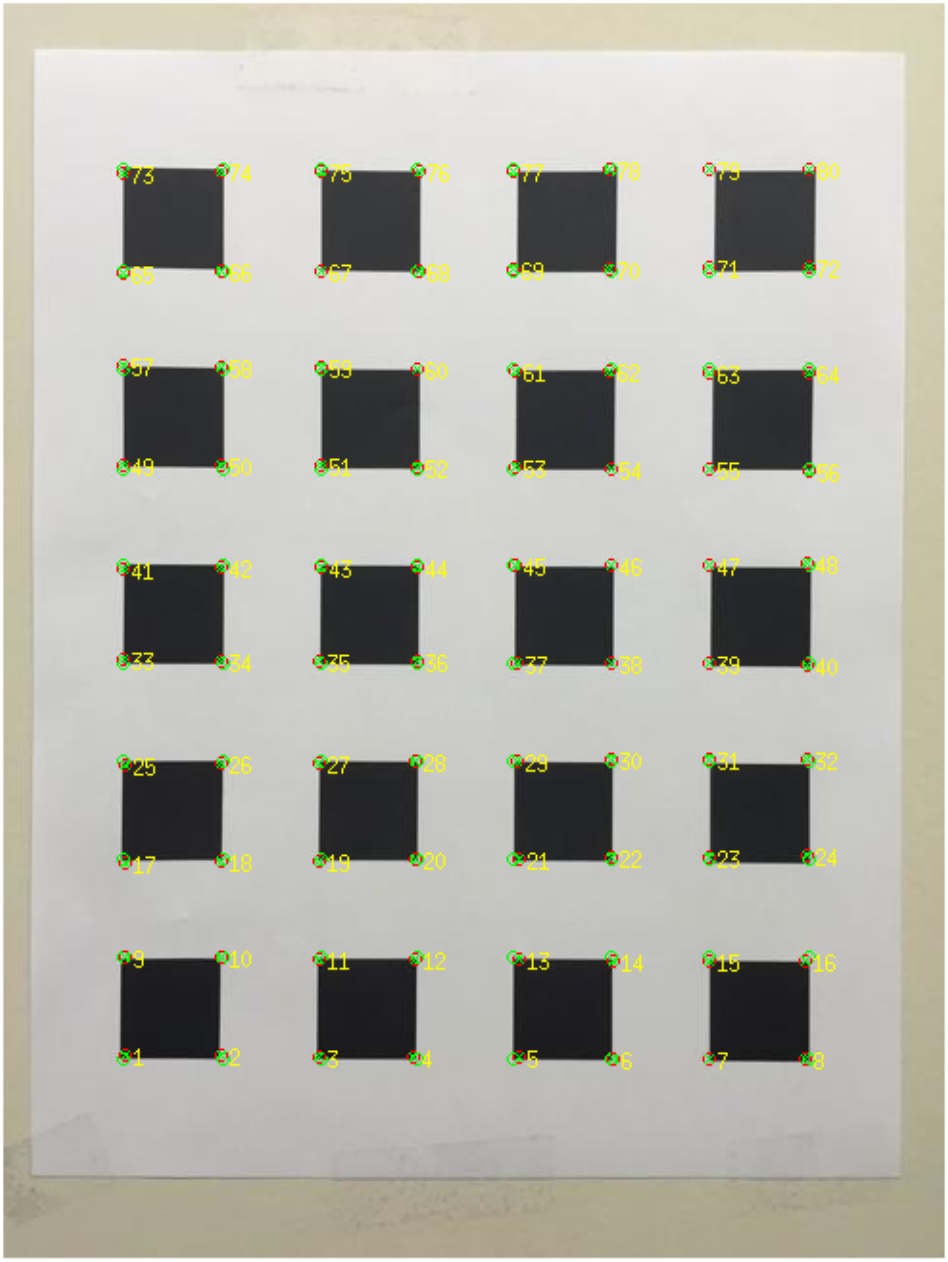

Fig 37. Re-project Corner on Pic21.jpg back to Pic1.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

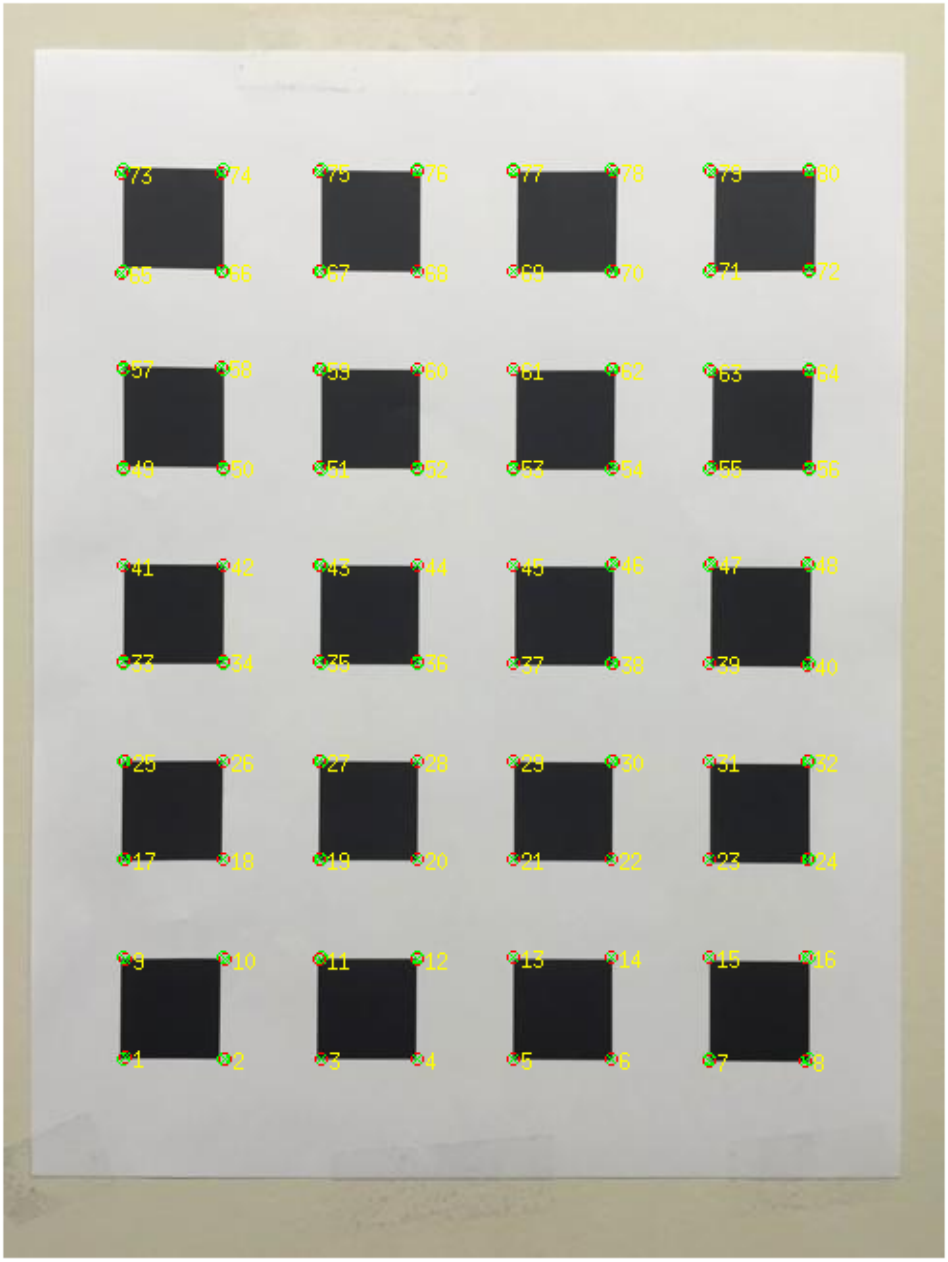

Fig 38. Re-project Corner on Pic22.jpg back to Pic1.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

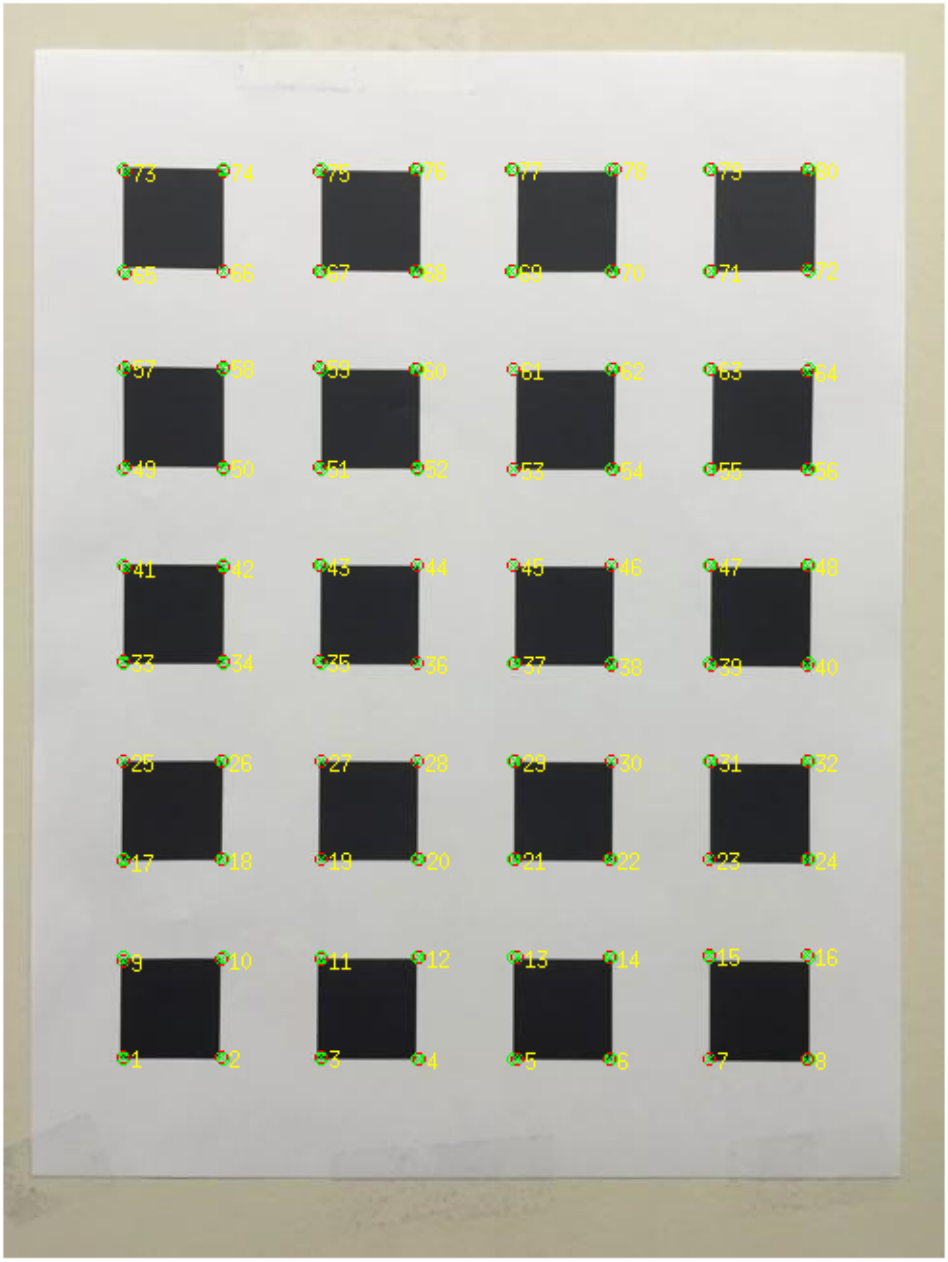

Fig 39. Re-project Corner on Pic23.jpg back to Pic1.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

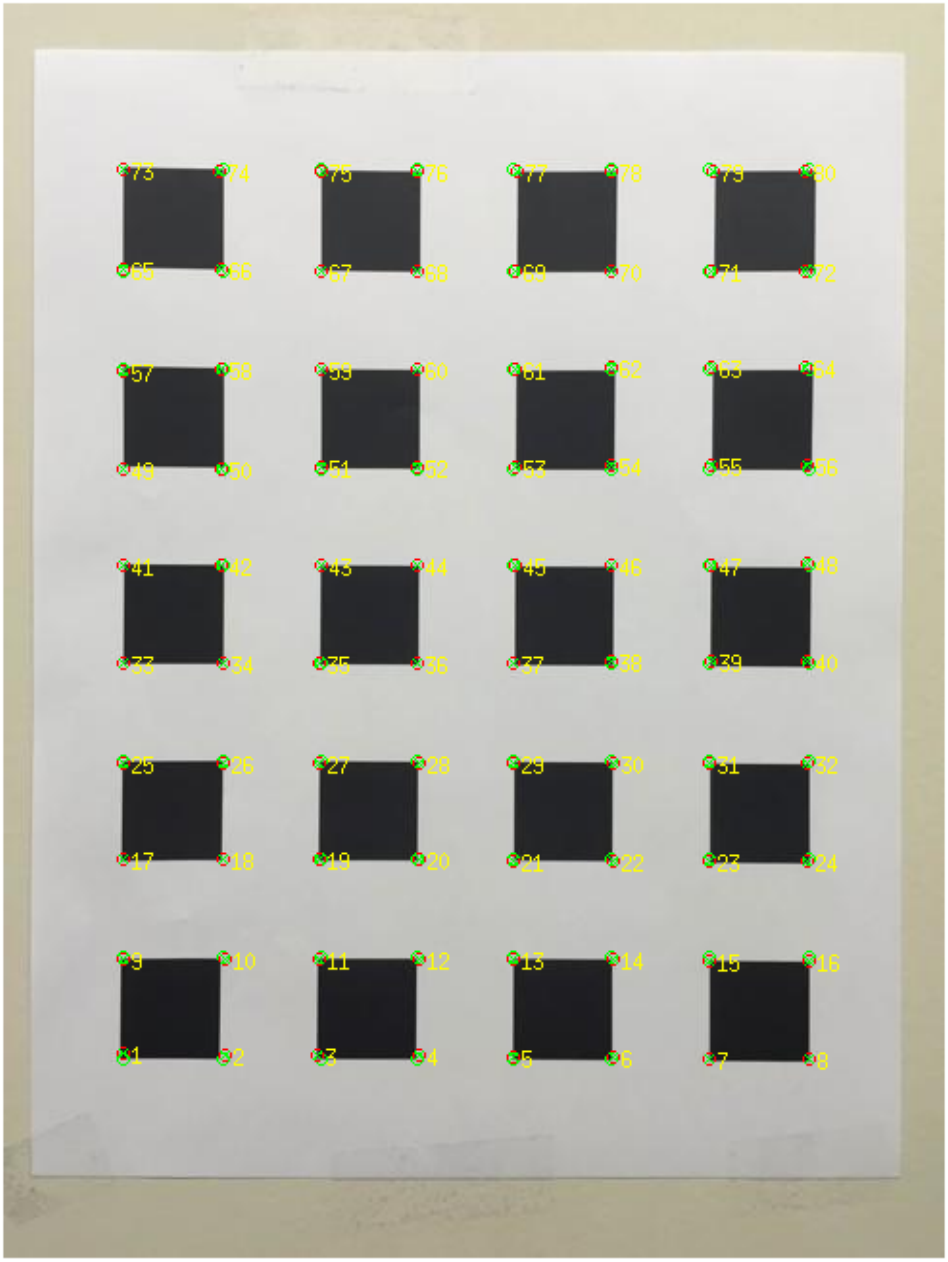

Fig 40. Re-project Corner on Pic24.jpg back to Pic1.jpg. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

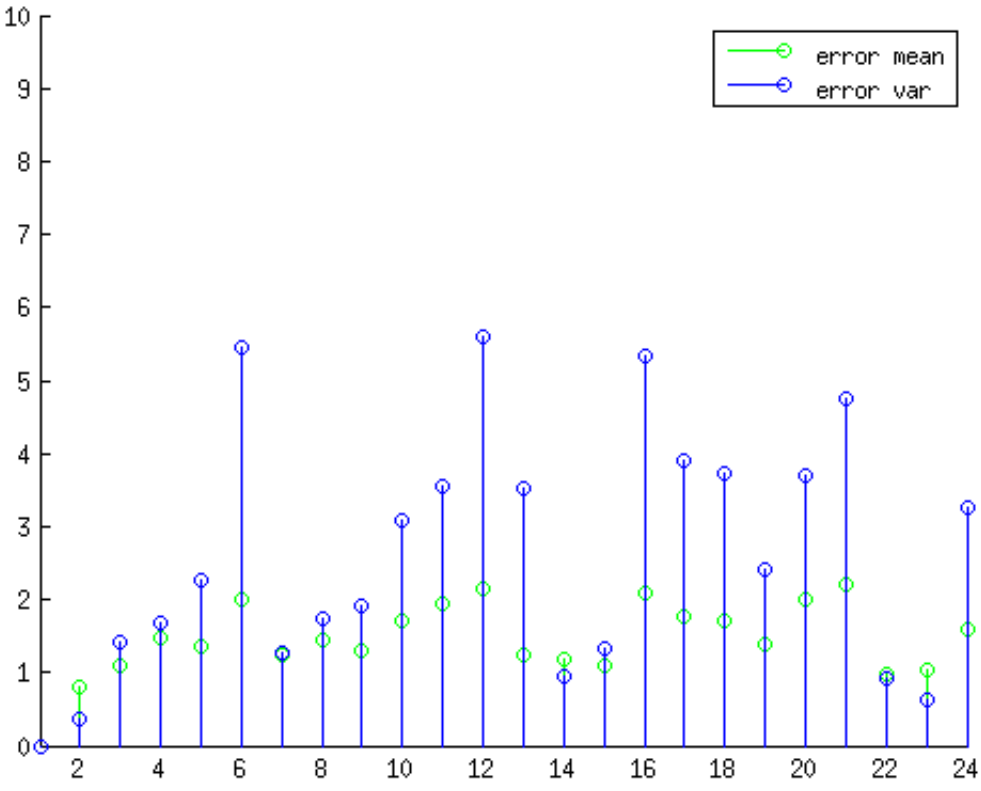

Fig 41. Quantitatively demonstrate the re-projection error.

MyDataset Conclusion: As all corner in all images were detected pretty accurately, we can see that re-projection is also accurate with average error between 1-2 pixels.

6 Improvement Using Levenberg-Marquardt Optimization

In general all corners are detected quite accurately, thus it is very difficult to find the image which the improvement could be seen visually. Luckily, for the `Dataset1` we have two images: `Pic10.jpg` and `Pic28.jpg` that some corners are completely wrong so we can observe the significant improvement after LM optimization.

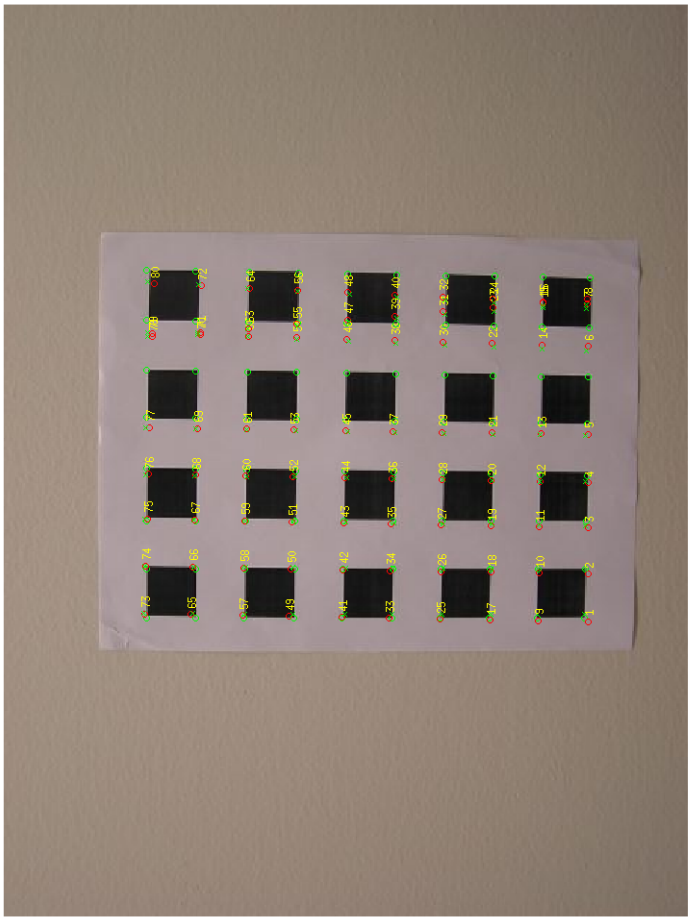

Fig 42. Significant improvement for the Pic10.jpg whose corners were wrong. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

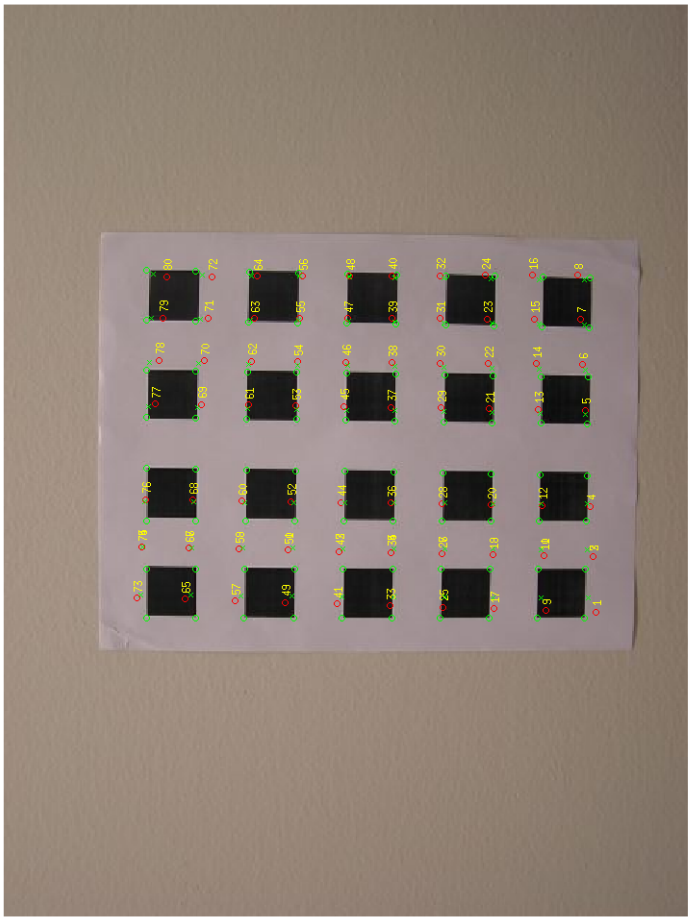

Fig 43. Significant improvement for the Pic28.jpg whose corners were wrong. Green Circle is the original detected corner, red circle is re-projected corners without refinement, green cross is the re-projected corners with refinement.

Quantitatively Demonstrate Significant Error Reduction

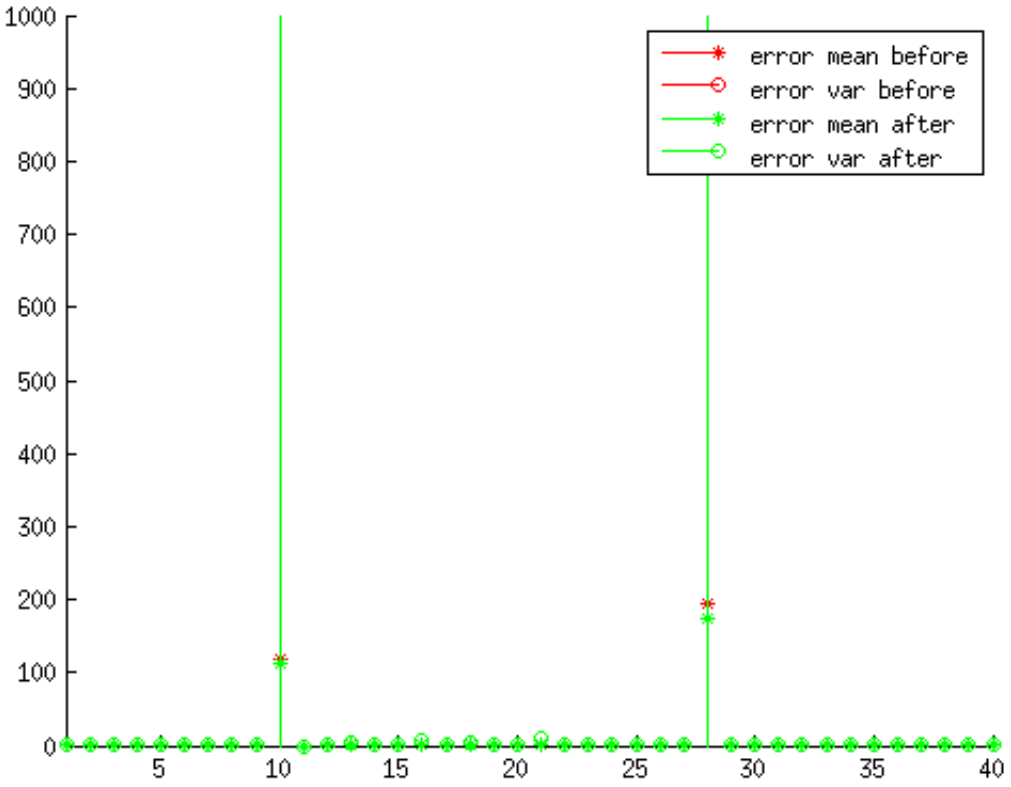

Fig 44. On the plot above we can easily see that the mean of error in euclidean distance decreased significantly after we apply LM algorithm on `Pic11.jpg` and `Pic28.jpg`.

6.1 Intrinsic and Extrinsic Parameters

Intrinsic Parameters for Dataset1

$$K = \begin{bmatrix} 728.4269 & 0.8973 & 317.7210 \\ 0 & 726.5993 & 237.4868 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

`Note that` x `and` y `axis is highly dependent on how we define` x `and` y `axis in the plane.`

Intrinsic Parameters for My Own Dataset

$$K_{iPhone6} = \begin{bmatrix} 592.5192 & 0.0763 & 251.4654 \\ 0 & 595.0980 & 329.9309 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

`Note that` x `and` y `axis is highly dependent on how we define` x `and` y `axis in the plane.`

Extrinsic Parameters for My Own Dataset The ground truth image was taken orthogonal to the calibration pattern plane, the distance between the calibration pattern and the camera was 30 cm.

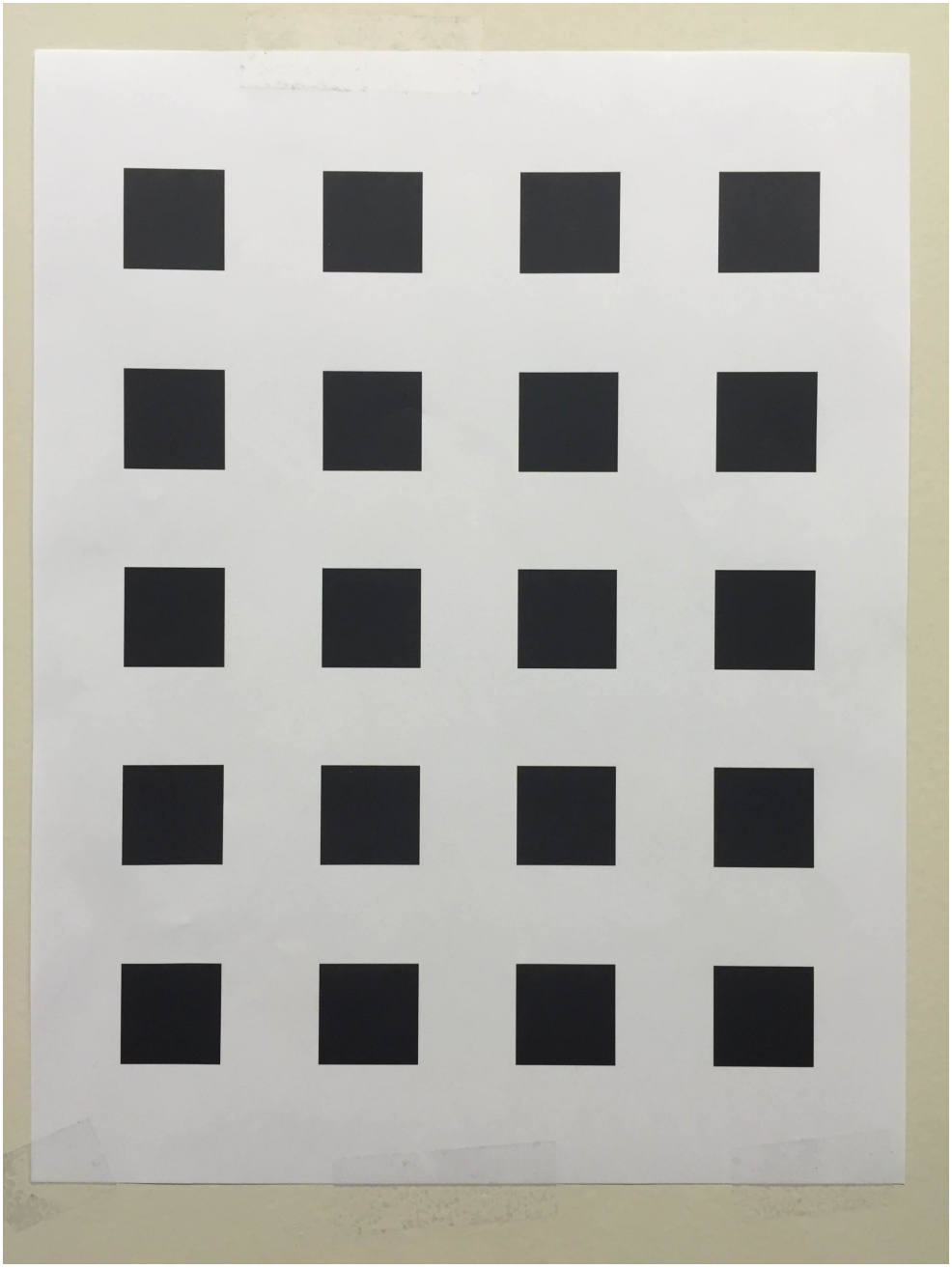

Fig 45. The Ground-Truth Image

List of Several Extrinsic Parameters:

$$My\ Dataset:\ Pic1\ (ground\ truth),\ R = 10^{-05} * \begin{bmatrix} -0.6326 & -0.0001 & -0.0000 \\ -0.0000 & 0.6348 & 0.0000 \\ 0.0015 & 0.0010 & -0.0000 \end{bmatrix},\ \vec{t} = \begin{bmatrix} 0.0006 \\ -0.0007 \\ -0.0018 \end{bmatrix}$$

As there was no rotation for the ground truth image, this result is actually very good. Thus after comparison with ground truth we can assume the extrinsic estimation is accurate enough.

$$My\ Dataset:\ Pic20,\ R = 10^{-05} * \begin{bmatrix} -0.5742 & -0.1608 & -0.0000 \\ -0.1504 & 0.5804 & -0.0000 \\ 0.1164 & -0.0373 & -0.0000 \end{bmatrix},\ \vec{t} = \begin{bmatrix} 0.0007 \\ -0.0005 \\ -0.0020 \end{bmatrix}$$

$$My\ Dataset:\ Pic21,\ R = 10^{-05} * \begin{bmatrix} 0.4887 & -0.1265 & -0.0000 \\ -0.1256 & -0.4849 & -0.0000 \\ -0.0036 & 0.0228 & -0.0000 \end{bmatrix},\ \vec{t} = \begin{bmatrix} -0.0003 \\ 0.0007 \\ 0.0017 \end{bmatrix}$$

$$My\ Dataset:\ Pic22,\ R = 10^{-05} * \begin{bmatrix} -0.5673 & -0.0495 & 0.0000 \\ -0.0512 & 0.5762 & -0.0000 \\ -0.0971 & -0.0096 & -0.0000 \end{bmatrix},\ \vec{t} = \begin{bmatrix} 0.0006 \\ -0.0006 \\ -0.0019 \end{bmatrix}$$

$$My\ Dataset:\ Pic23,\ R = 10^{-05} * \begin{bmatrix} -0.5575 & 0.0446 & 0.0000 \\ 0.0484 & 0.5656 & 0.0000 \\ -0.0880 & 0.0238 & -0.0000 \end{bmatrix},\ \vec{t} = \begin{bmatrix} 0.0005 \\ -0.0007 \\ -0.0017 \end{bmatrix}$$

$$My\ Dataset:\ Pic24,\ R = 10^{-05} * \begin{bmatrix} 0.4954 & 0.1090 & -0.0000 \\ 0.1210 & -0.5077 & 0.0000 \\ -0.1117 & -0.0659 & -0.0000 \end{bmatrix},\ \vec{t} = \begin{bmatrix} -0.0007 \\ 0.0006 \\ 0.0020 \end{bmatrix}$$

7 Appendix: `Matlab` Code Used in This Assignment

Please refer to the `.zip` file.