FALL 22 ECE 661 – COMPUTER VISION

Homework 8

Sahithi Kodali – 34789866
kodali1@purdue.edu

A. THEORY QUESTION

In Lecture 20, we showed that the image of the Absolute Conic Ω_∞ is given by $\omega = K^{-T}K^{-1}$. As you know, the Absolute Conic resides in the plane π_∞ at infinity. Does the derivation we went through in Lecture 20 mean that you can actually see ω in a camera image? Give reasons for both 'yes' and 'no' answers. Also, explain in your own words the role played by this result in camera calibration.

The image pixels of the absolute conic *w* are actually imaginary since its value $K^{-T}K^{-1}$ is semi definite and hence not possible to plot these pixels as a visionary image. Hence, we can say that we cannot actually see *w* in the camera image. However, we can say that it exists in the non-visible plane π_∞ at infinity and has circular points. It is hence useful to determine the matrix K intrinsic parameters of the camera which is very important in camera calibration as shown in Zhang's algorithm.

B. PROGRAMMING QUESTION

The programming task involves implementing Zhang's algorithm for the given dataset of 40 images of checkerboard pattern and an own dataset of images of this checkerboard stuck to a wall.

Zhang's Algorithm:

It involves two main steps of corner detection and camera calibration as below.

1. Corner Detection

The first step is to detect corners using Canny Edge Detector from OpenCV. The images are converted into grayscale images and, a Gaussian (3x3) filter is used to smoothen the given dataset and a (5x5) filter to smoothen the own dataset.

This is followed by extracting lines from edges using Hough transform from OpenCV. The parameters are set by trails and error which can be seen in the source code comments. The lines extracted are separated into vertical and horizontal lines based on the slope/angle threshold they subtend to y-axis. Many lines can be detected by Hough transform; however, we require only 10 horizontal lines and 8 vertical lines for a checkboard edge lines. To resolve this, a distance-based compression is used where the distance between lines is sorted by a threshold (set my trial and error as shown in code comments) and the lines less than the threshold are considered as a single line which is the average of the group of lines. Thus, we get 8 vertical lines and 10 horizontal lines.

Using the lines, we compute the intersection points and plot them along with their labels. These intersection points are further used in the camera calibration.

2. Camera Calibration

In the camera calibration step, we first compute the correspondence between the original image that has the checkboard grid coordinates and all the Homographies for transforming the original world image to the views of image coordinates of given dataset and the own dataset. The Homographies are used to compute w from which the intrinsic matrix K and the extrinsic matrices R,t are computed. The matrices are further refined using Levenberg-Marquardt (LM) optimization. The radial distortion is also considered for the parameters and conditioned as the final step.

Assuming that the plane the world pattern lies is z = 0 and x is a salient point on this plane, the image coordinates are given as,

$$x_i = Hx_w = K[R|t]\begin{bmatrix} x \\ y \\ 0 \\ w \end{bmatrix}$$

Here, K is the intrinsic matrix; R and t are the rotation and translation matrices and are extrinsic.

2.1 Calculation of Intrinsic parameter K

In the above H = [h_1 h_2 h_3] and the two circular points I and J on the absolute conic (w) are given as, I = [1, I, 0] ^{T} and J = [1, -I, 0] ^{T}. Their images can be given as HI = h_1 + ih_2 and HJ = h_1 – ih_2. Since these points lie on *w*,

$$(HI)^T w (HI) = (h_1 + ih_2)^T w (h_1 + ih_2) = 0$$

$$(HJ)^T w (HJ) = (h_1 - ih_2)^T w (h_1 - ih_2) = 0$$

Solving the above we get the equations (h_1^T w h_1) – (h_2^T w h_2) = 0 and h_1^T w h_2 = 0, where

$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \text{ and } \quad h_1 = [h_{11}\ h_{21}\ h_{31}], \quad h_2 = [h_{12}\ h_{22}\ h_{32}]$$

In the above equations, *w* can be flattened to represent as below as shown in Zhang algorithm. Since *w* is symmetric it has only 6 parameters.

$$Vb = \begin{bmatrix} v_{12}^T \\ v_{11}^T - v_{22}^T \end{bmatrix} b = 0$$

$$\text{where, } v_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix} \quad b = \begin{bmatrix} w_{11} \\ w_{12} \\ w_{22} \\ w_{13} \\ w_{23} \\ w_{33} \end{bmatrix}$$

For *n* images, these equations are stacked to compute *b* using linear least squares method and optimized using LM optimization, from which the matrix *w* can be attained. From *w* we can compute matrix K since $w = K^{-T}K^{-1}$,

where, $K = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$ and the 5 unknown parameters are calculated as,

$$y_0 = \frac{w_{12}w_{13} - w_{11}w_{23}}{w_{11}w_{22} - {w_{12}}^2}$$

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

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

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

$$x_0 = \frac{sy_0}{\alpha_y} - \frac{w_{13}{\alpha_x}^2}{\lambda}$$

$$\lambda = w_{33} - \frac{{w_{13}}^2 + y_0(w_{12}w_{13} - w_{11}w_{23})}{w_{11}}$$

2.2 Calculation of Extrinsic parameters R and t

From above, $H = K[R|t]$ => K^{-1} $[h_1\ h_2\ h_3] = [r_1\ r_2\ t]$. Since the rotation matrix need to be orthogonal it is multiplied by a scaling factor $\xi = 1/\| K^{-1} h_1\|^{-1}$ giving the parameters,

$$r_1 = \xi\ K^{-1} h_1$$

$$r_2 = \xi\ K^{-1} h_2$$

$$r_3 = r_1 \text{ X } r_2$$

$$t = \xi\ K^{-1} h_3$$

To refine the parameters using LM optimization, the 9 parameters in $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ needs to be converted to a 3 parameters representation (as it has 3 DoF) using Rodrigues representation given as W.

$$W = \frac{\varphi}{2\sin(\varphi)} \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} \text{ where } \varphi = \cos^{-1}\left(\frac{trace(R)-1}{2}\right)$$

The W obtained is transformed to R as below,

$$R = e^{[w]_x} = I_{3X3} + \frac{\sin(\varphi)}{\varphi}[w]_x + \frac{1-\cos(\varphi)}{\varphi}[w]_x{}^2$$

$$\text{where, } \varphi = \left||w|\right| \; and \; [w]_x = \begin{bmatrix} 0 & -w_z & -w_y \\ -w_z & 0 & -w_x \\ -w_y & -w_x & 0 \end{bmatrix}$$

2.3 Parameter refinement using LM

The matrices K, R, t are refined using the LM optimisation by computing the cost function (d^2_{goem}) of geometric distance than needs to be minimized.

$$d^2_{goem} = \| X - f(p) \|2 = \sum_i \sum_j || \; x_{ij} - x'_{ij} || \text{ where } x_{ij} = K[R_i | \; t_i] x_{mj}$$

In the above, x_{ij} is the actual image point of the j[th] salient point on calibration world pattern x_{mj} and x'_{ij} is the projected image point for i[th] camera view.

2.4 Radial Distortion

To get rid of the assumption that the camera is a pinhole camera when focal length is long, we also need to consider the case of using camera with a short focal length as used in this scenario which is said as considering the radial distortion. The x_{ij} projected points after derived are modified using two radial distortion parameters k1 and k2 giving the coordinates,

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

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

where, $r^2 = (x` - x_0)^2 + (y` - y_0)^2$ and, (x`, y`) are the projection points while (x_0, y_0) is the principal point of the image.

The parameters thus refined and optimized using LM cost function give better camera parameters which are further discussed.

C. RESULTS & OBSERVATIONS

1. Given Dataset

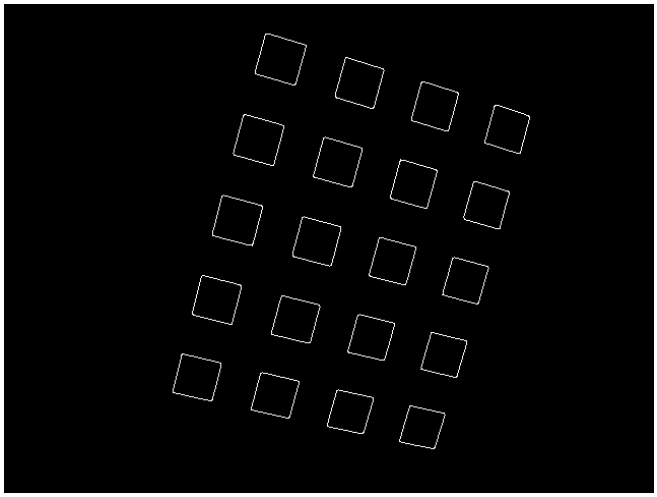


Figure 1 – Edges using canny detector for image 16

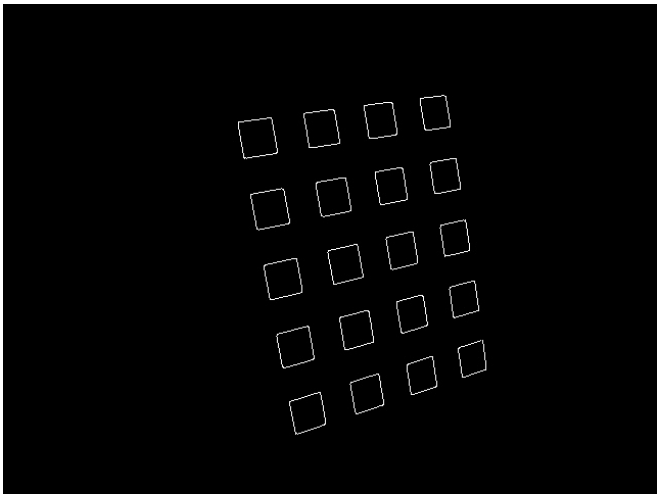


Figure 2 – Edges using canny detector for image 22

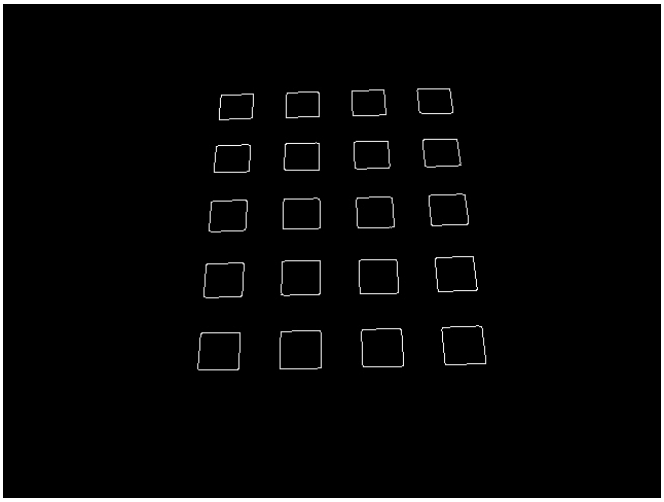


Figure 3 – Edges using canny detector for image 37

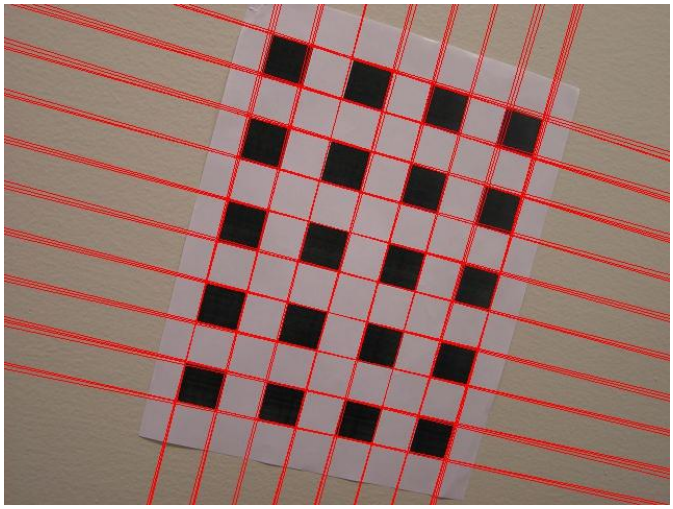


Figure 4 – Hough lines for image 16

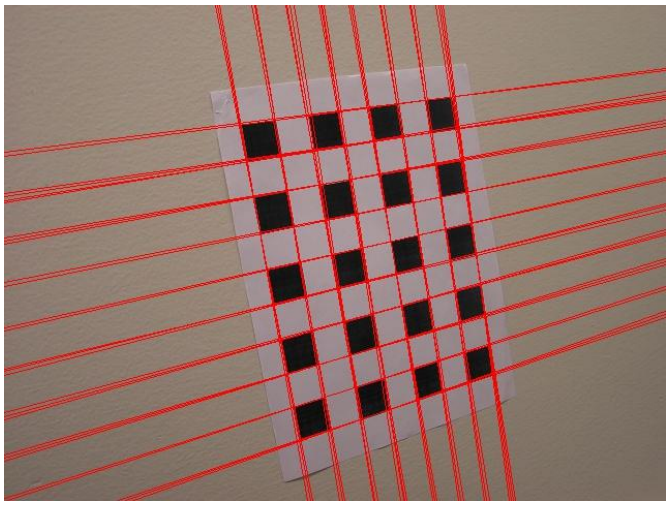


Figure 5 – Hough lines for image 22

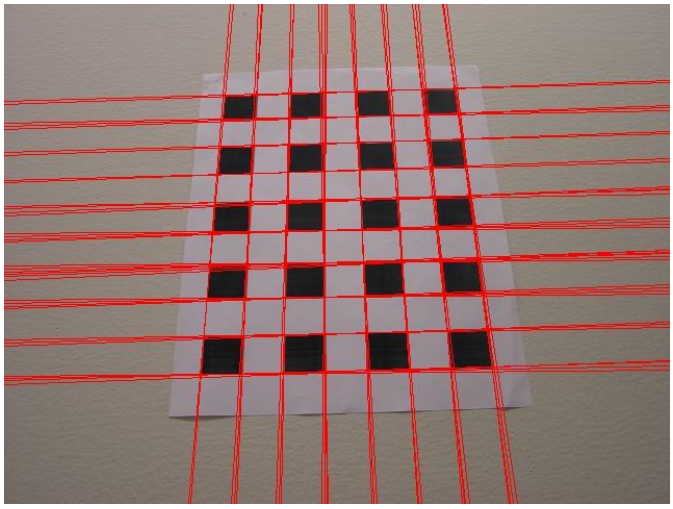


Figure 6 – Hough lines for image 37

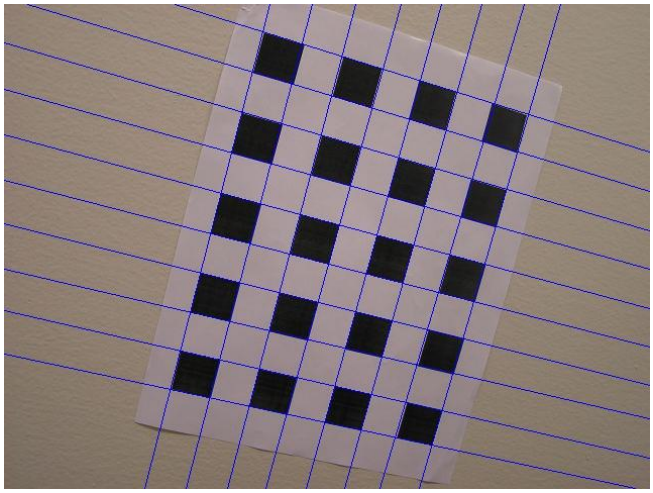


Figure 7 – Filtered Hough lines for image 4

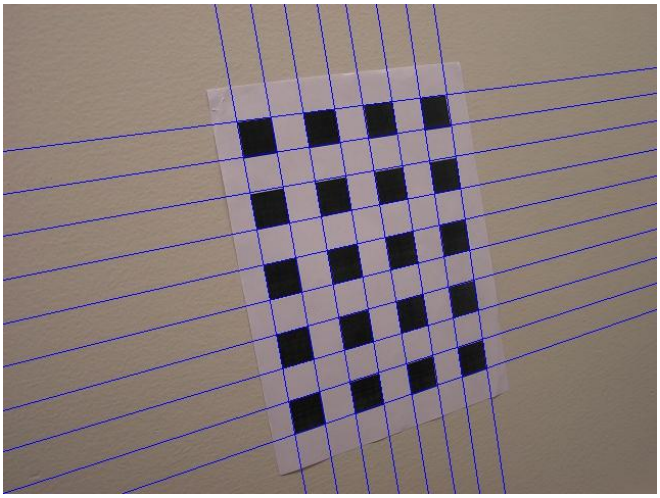


Figure 8 – Filtered Hough lines for image 22

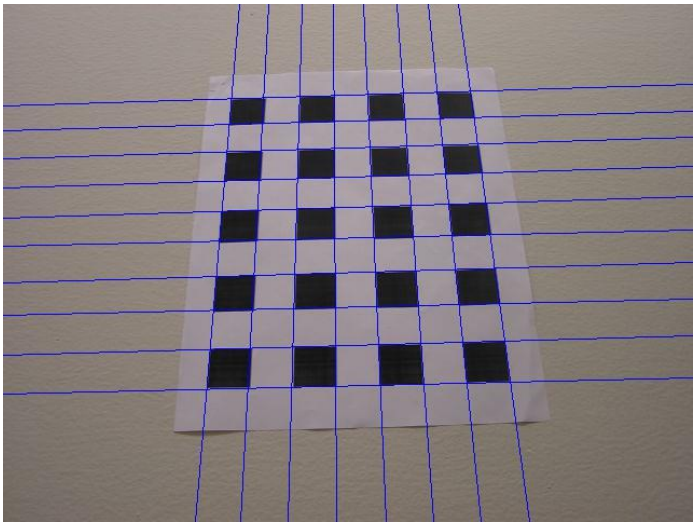


Figure 9 – Filtered Hough lines for image 37

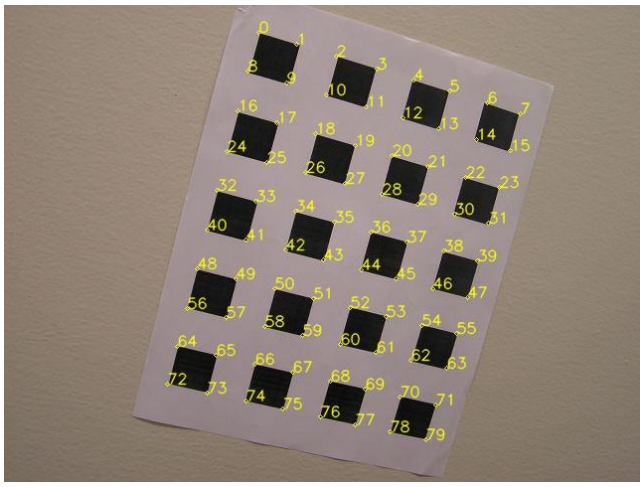


Figure 10 – Intersection points for image 16

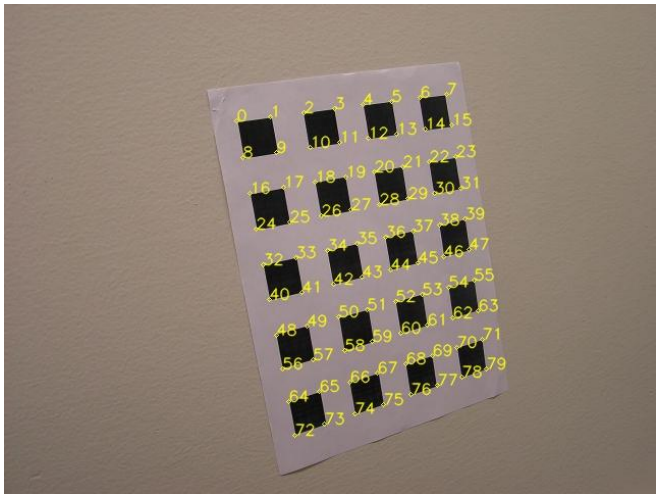


Figure 11 – Intersection points for image 22

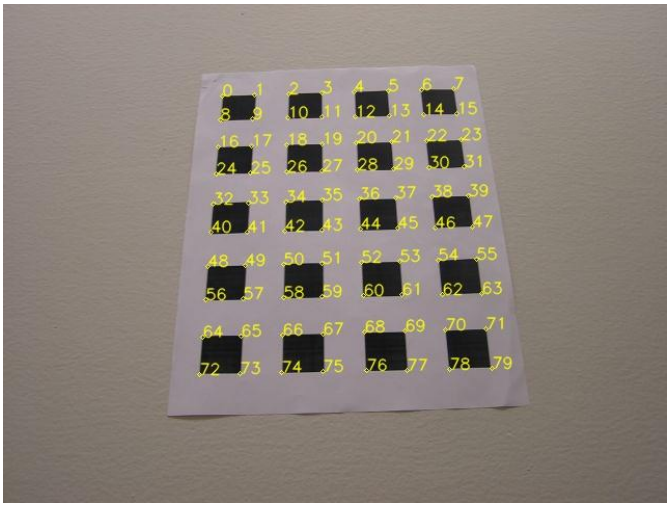


Figure 12 – Intersection points for image 37

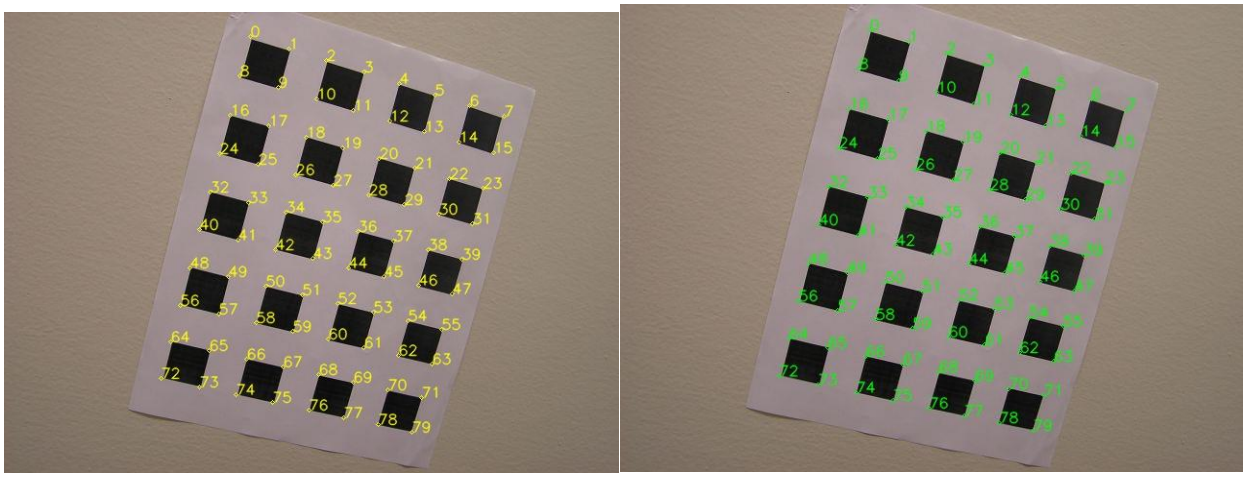


Figure 13 – Reprojected corners of image 16 before(yellow) and after using LM optimization

Figure 14 – Reprojected corners of image 37 before(yellow) and after using LM optimization

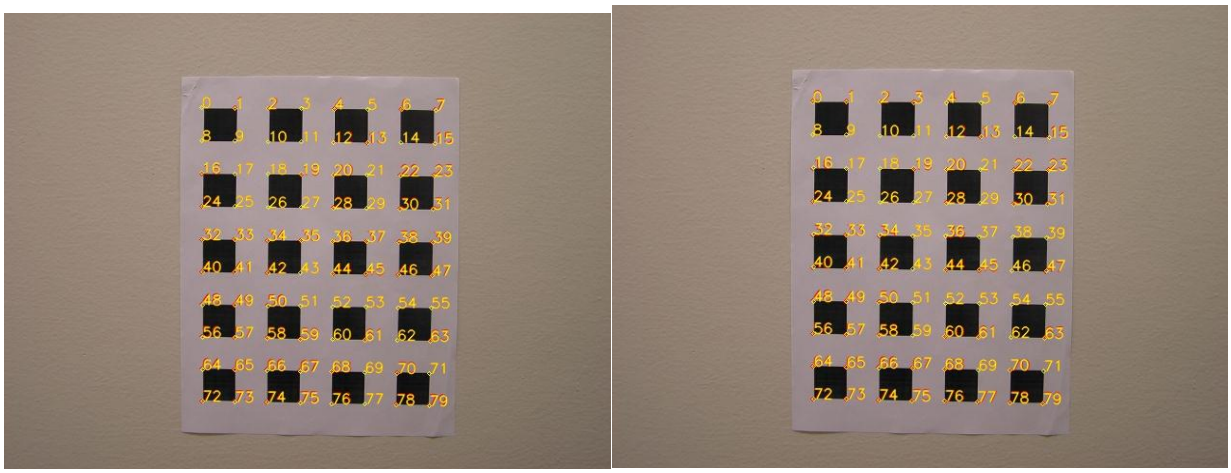


Figure 15 – Corners of image 16 projected on fixed image before(left) and after using LM

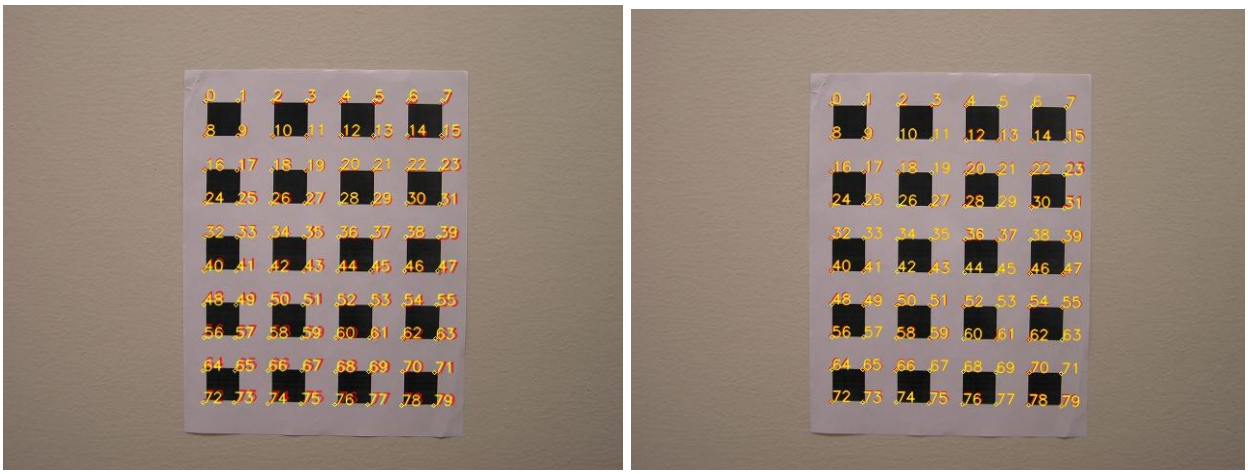


Figure 16 – Corners of image 37 projected on fixed image before(left) and after using LM

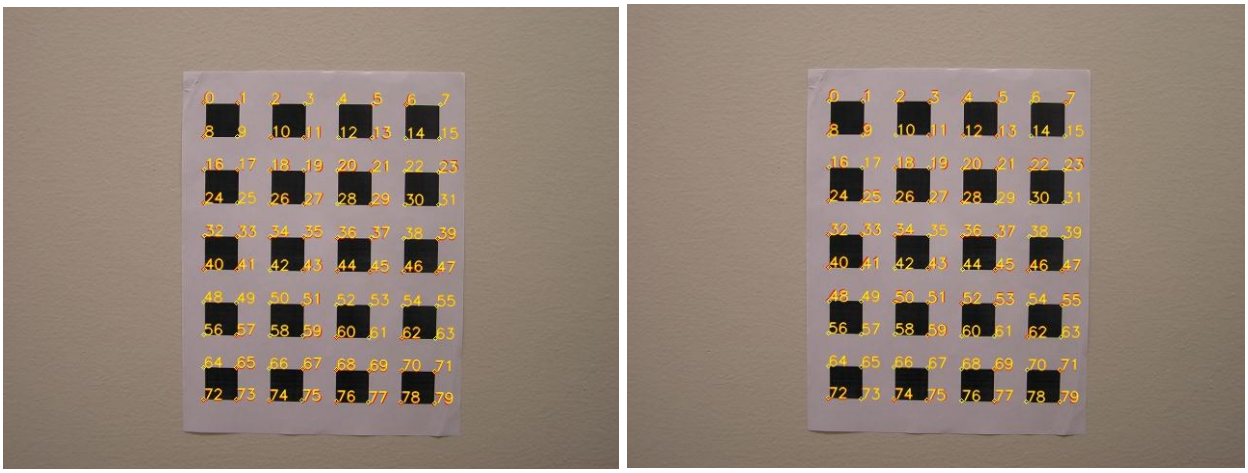


Figure 17 – Corners of image 22 on fixed image before and after radial distortion (Improved projection can be clearly seen in 48-55 labels)

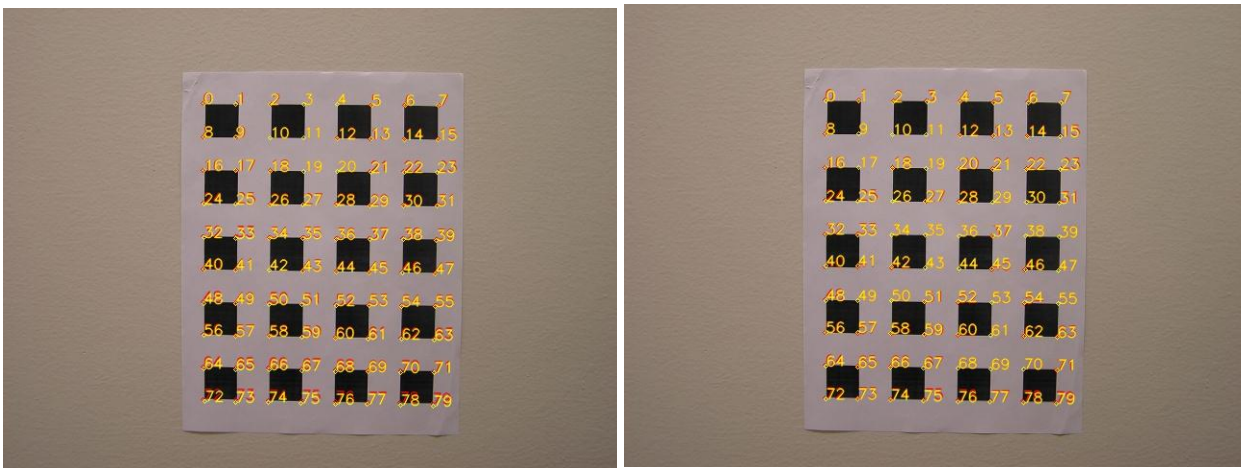


Figure 18 – Corners of image 35 on fixed image before and after radial distortion (Improved projection can be clearly seen in 64-79 labels)

Estimates of K, [R | t] with and without refinement using LM and radial distortion:

Intrinsic camera parameter K:

$$K = \begin{bmatrix} 715.26 & 1.352 & 321.46 \\ 0 & 713.20 & 241.27 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K\_LMrefined = \begin{bmatrix} 721.28 & 2.213 & 321.60 \\ 0 & 719.20 & 241.98 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K\_LMrefined\_radial = \begin{bmatrix} 726.80 & 2.241 & 320.61 \\ 0 & 724.86 & 243.84 \\ 0 & 0 & 1 \end{bmatrix}$$

We can observe the improvement in the K matrix as it is refined and incorporated with radial distortion.

Extrinsic camera parameters [R|t]:

For image 22:

$$[R|t] = \begin{bmatrix} 0.875 & 0.162 & -0.456 & -25.444 \\ -0.201 & 0.978 & -0.039 & -34.928 \\ 0.439 & 0.126 & 0.889 & 196.225 \end{bmatrix}$$

$$[R|t]\_LMrefined = \begin{bmatrix} 0.878 & 0.163 & -0.447 & -25.53 \\ -0.200 & 0.979 & -0.035 & -35.33 \\ 0.432 & 0.120 & 0.893 & 199.696 \end{bmatrix}$$

$$[R|t]\_LMrefined\_radial = \begin{bmatrix} 0.880 & 0.164 & -0.445 & -24.996 \\ -0.201 & 0.978 & -0.036 & -35.848 \\ 0.430 & 0.121 & 0.894 & 200.411 \end{bmatrix}$$

For image 37:

$$[R|t] = \begin{bmatrix} 0.999 & 0.013 & 0.029 & -34.485 \\ -0.027 & 0.849 & 0.526 & -48.419 \\ -0.018 & -0.526 & 0.849 & 224.990 \end{bmatrix}$$

$$[R|t]\_LMrefined = \begin{bmatrix} 0.999 & 0.014 & 0.027 & -34.464 \\ -0.026 & 0.854 & 0.518 & -48.501 \\ -0.015 & -0.519 & 0.854 & 225.815 \end{bmatrix}$$

$$[R|t]\_LMrefined\_radial = \begin{bmatrix} 0.999 & 0.012 & 0.029 & -33.843 \\ -0.26 & 0.855 & 0.516 & -49.076 \\ -0.018 & -0.517 & 0.855 & 226.379 \end{bmatrix}$$

A fair amount of improvement can be observed in the R and t matrices above when refined and radial distortion is included. However, we can observe the changes to be minimal with radial distortion after LM refinement.

	Average error	***Variance***	***Max error***
Original parameters	1.02660864	0.5971901	5.80386693
LM refined only	0.61186022	0.12041649	2.68659332
LM refined with radial distortion	0.5419156	0.111816	2.6440906

Table 1: The overall projection error of all images with respect to world image. We can observe that the error metrics are reduced with LM and further by radial distortion.

	Original parameters	***LM refined only***	***LM refined with radial distortion***
Mean error_Img16	0.7273	0.6918	0.6499
Variance_Img16	0.1648	0.1533	0.1351
Mean error_Img37	1.2955	0.5556	0.5197
Variance_Img37	0.7038	0.0917	0.0958

Table 2: The projection metrics of image 16 and 37 with world pattern (refers to Figure 13 &14). We can observe the mean error and variance are reduced with LM and further by radial distortion.

	Average error	***Variance***	***Max error***
Original parameters	1.25411101	0.74864702	5.5105147
LM refined only	0.81832677	0.26427356	3.94949252
LM refined with radial distortion	0.75710019	0.17088915	2.6269424

Table 3: The overall projection error of all images with respect to fixed frame. We can observe that the error metrics are reduced with LM and further by radial distortion.

	Original parameters	***LM refined only***	***LM refined with radial distortion***
Mean error_Img16	1.1603	0.6349	0.6183
Variance_Img16	0.3624	0.1285	0.1070
Mean error_Img37	0.9375	0.6669	0.5904
Variance_Img37	0.1277	0.1046	0.0811

Table 4: The projection metrics of image 16 and 37 onto fixed image (refers to Figure 15 and 16). We can observe the mean error and variance are reduced with LM and further by radial distortion.

	Original parameters	***LM refined only***	***LM refined with radial distortion***
Mean error_Img22	0.8292	0.7909	0.4858
Variance_Img22	0.2039	0.1717	0.0561
Mean error_Img35	1.1249	0.6110	0.5348
Variance_Img35	0.3341	0.07681	0.0826

Table 5: The projection metrics of image 22 and 35 onto fixed image including radial distortion (refers to Figure 17 and 18). The radial distortion parameters computed are k1 = $-2.78654399e^{-7}$ *k2 =* $1.67409404e^{-12}$*. We can observe that the error metrics are reduced with LM and further by radial distortion.*

2. Own Dataset

22 images of the calibration pattern on the wall are considered as the own dataset. The lines may look broken due to them being thin, however in the actual images there appear to be no breaks for the images displayed below.

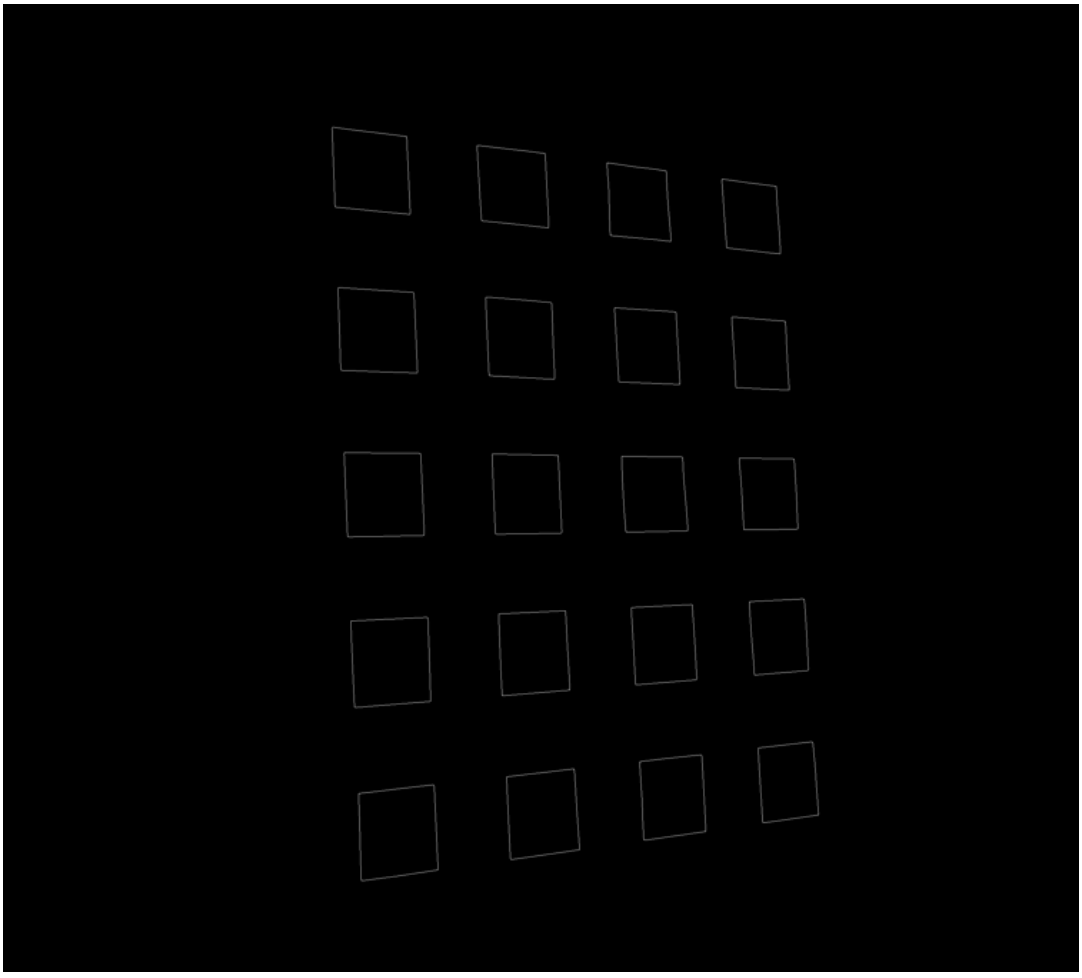


Figure 19 – Edges using canny detector for image 2

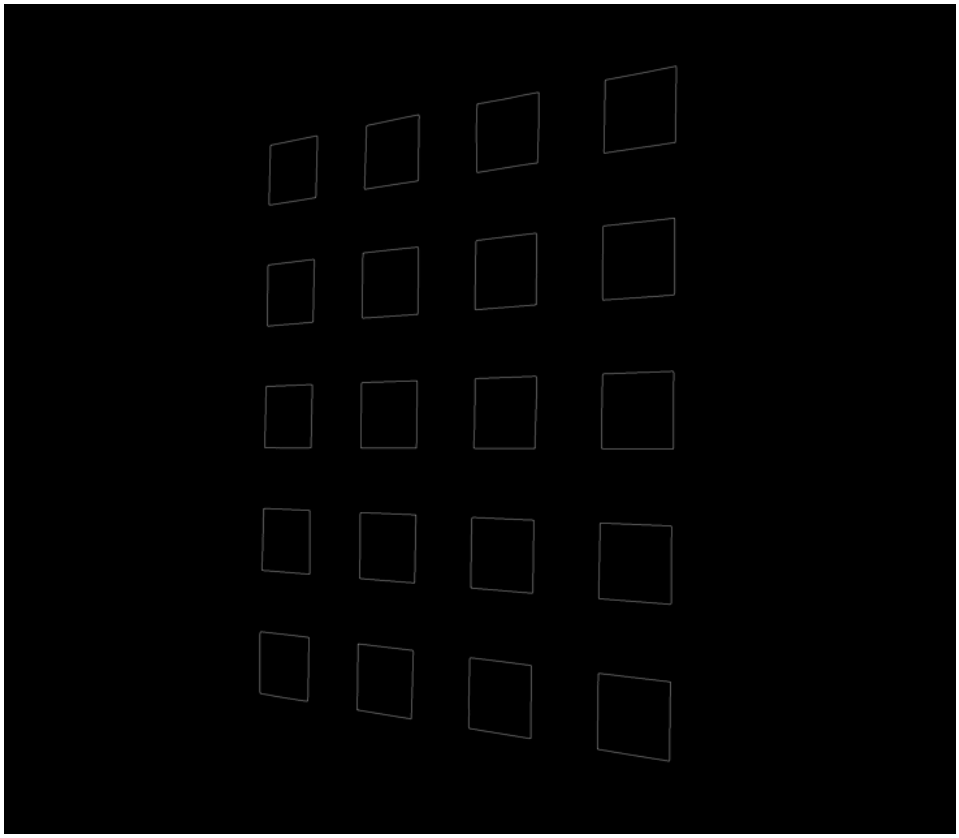


Figure 20 – Edges using canny detector for image 15

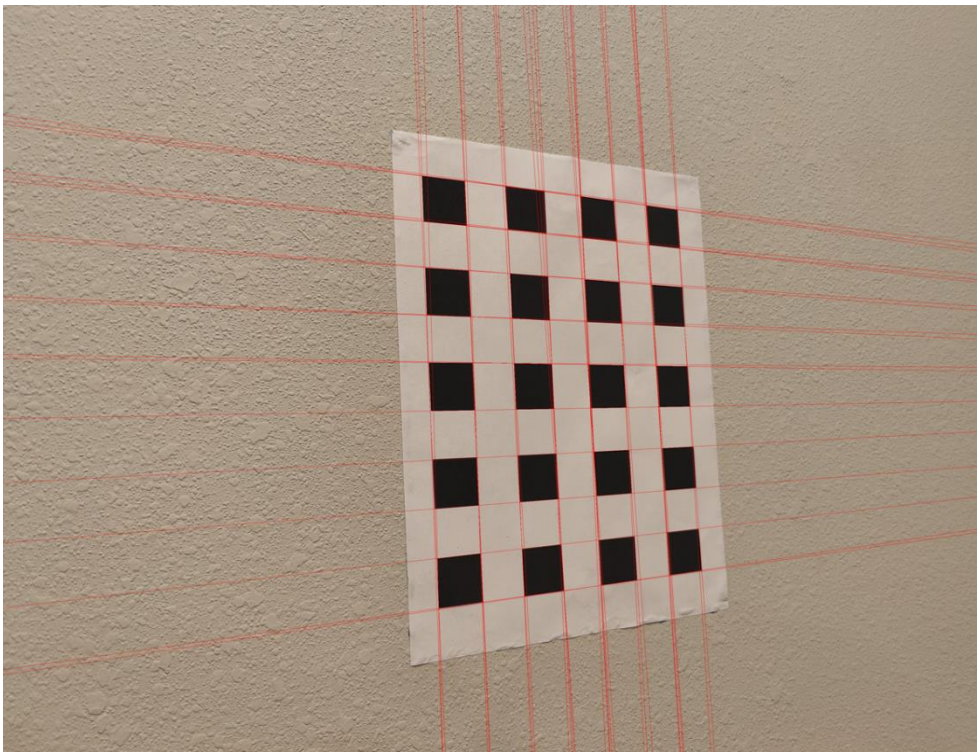


Figure 21 – Hough lines for image 2

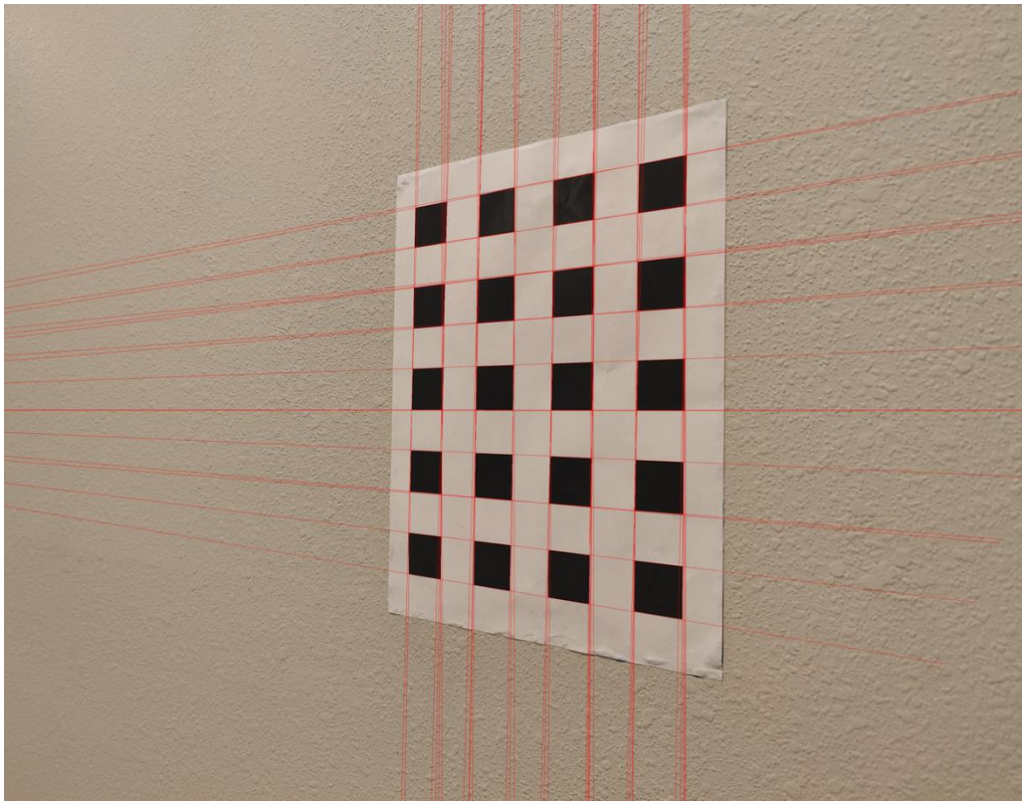


Figure 22 – Hough lines for image 15

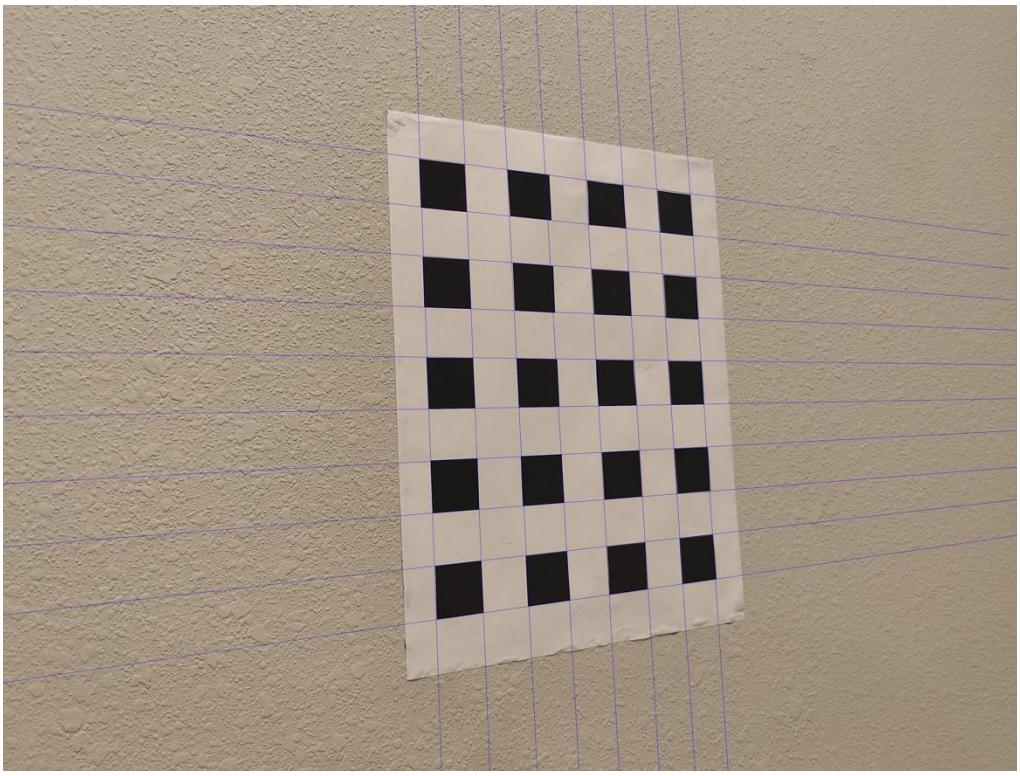


Figure 23 – Filtered Hough lines for image 2

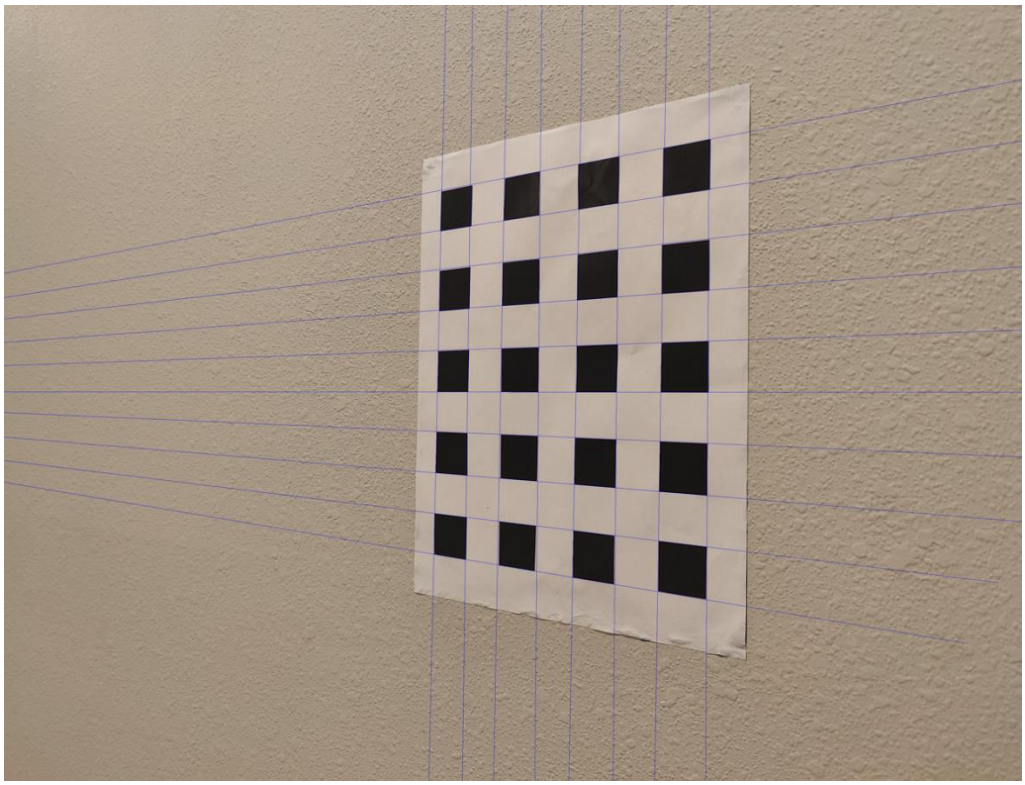


Figure 24 – Filtered Hough lines for image 15

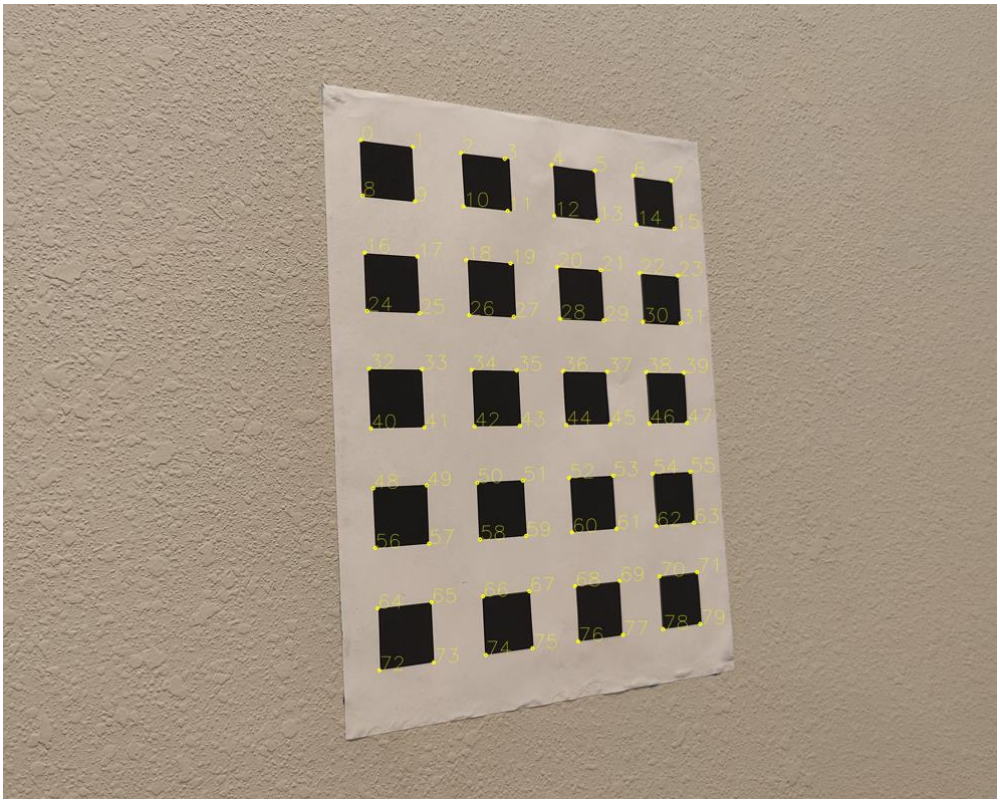


Figure 25 – Intersection points for image 2

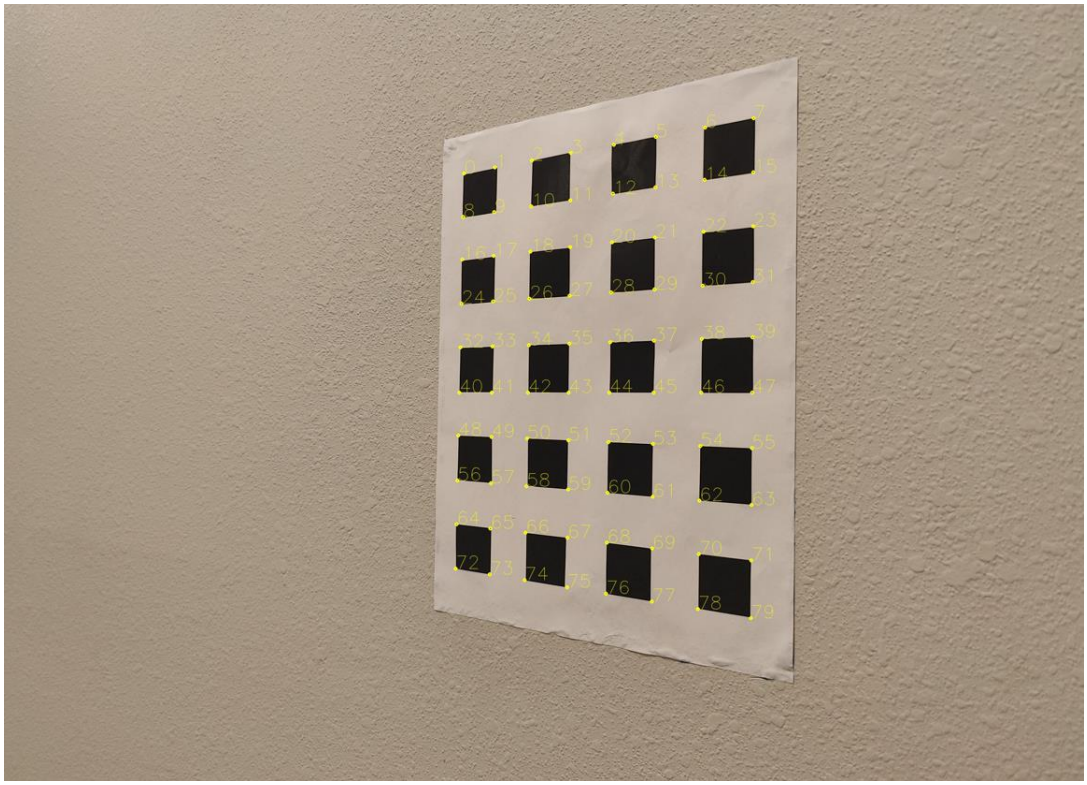
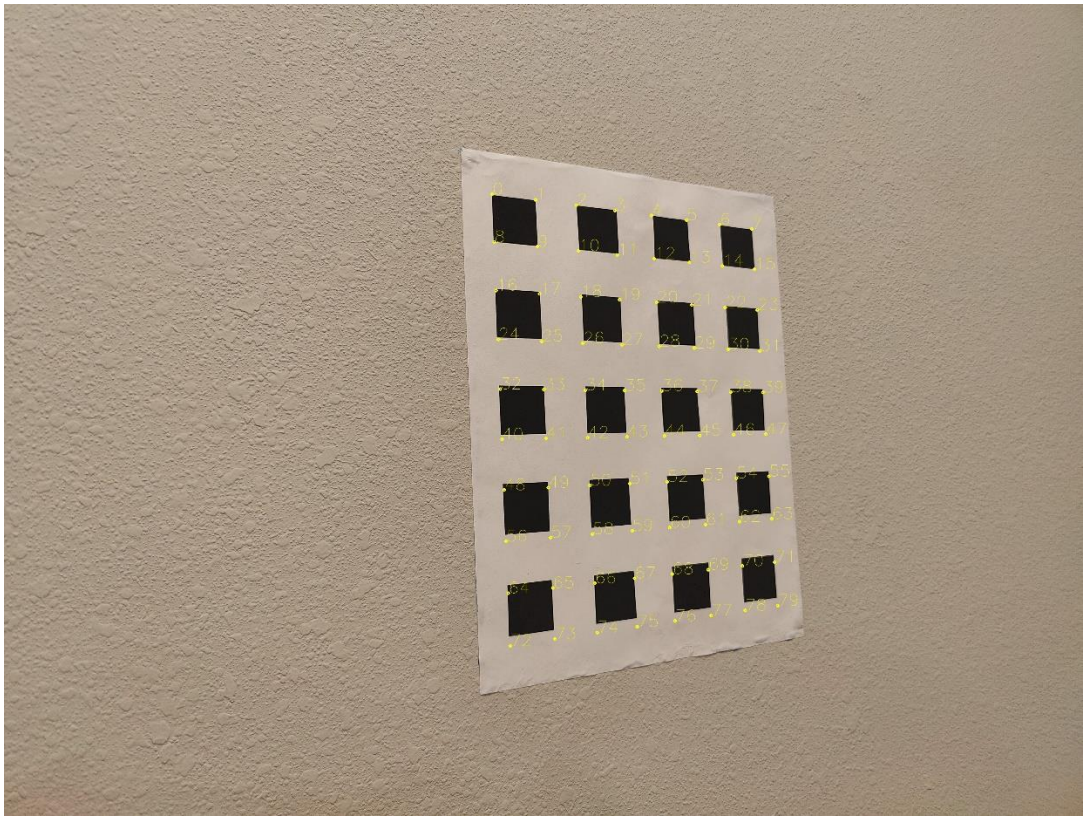


Figure 26 – Intersection points for image 15



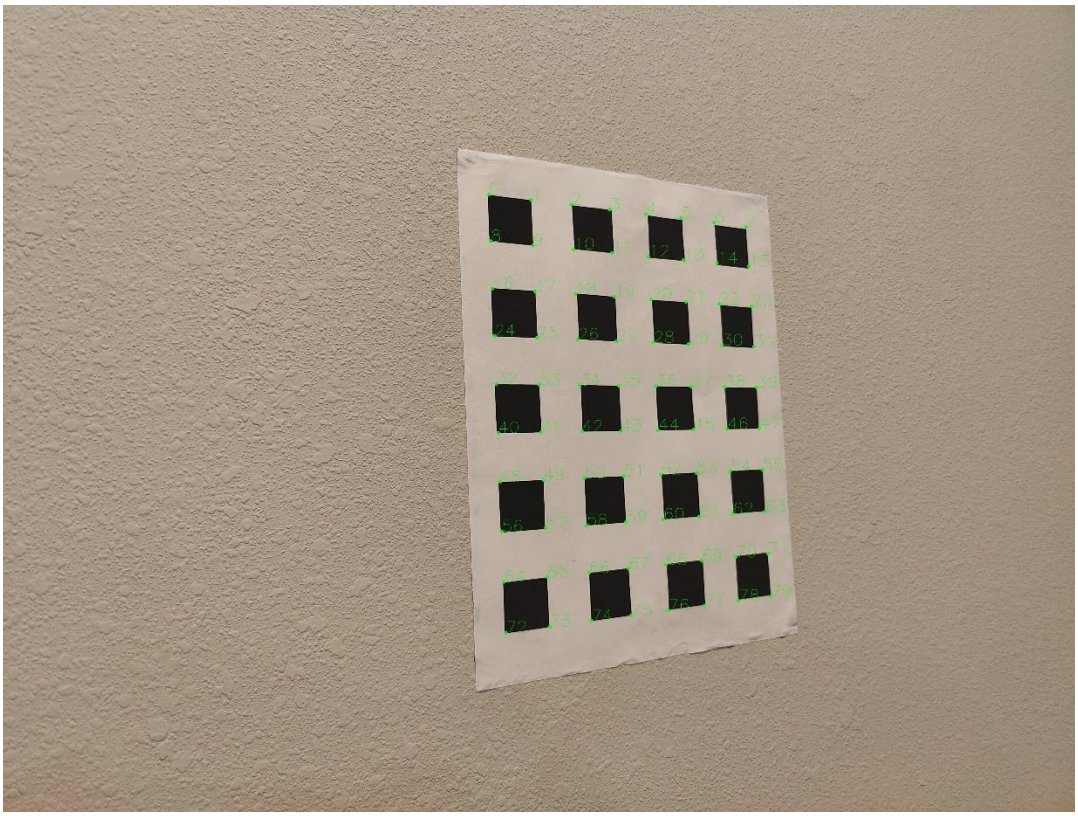
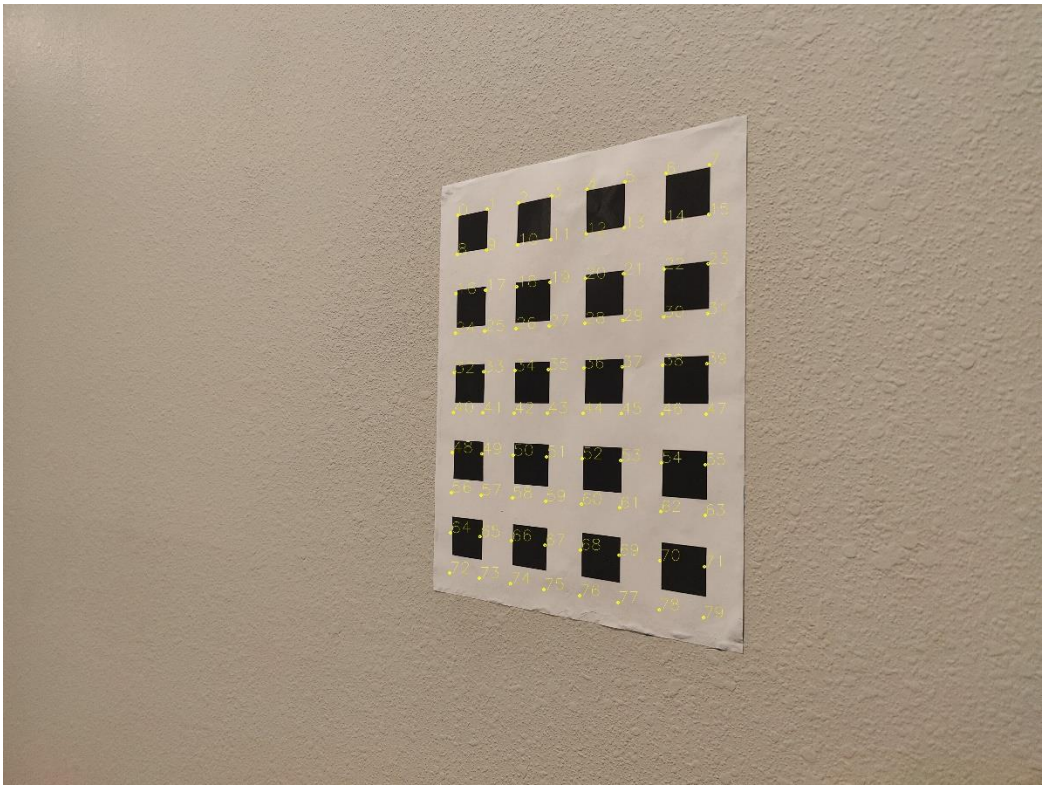


Figure 27 – Reprojected corners of image 2 before(yellow) and after using LM optimization (Difference can be clearly seen in the last row of pattern)



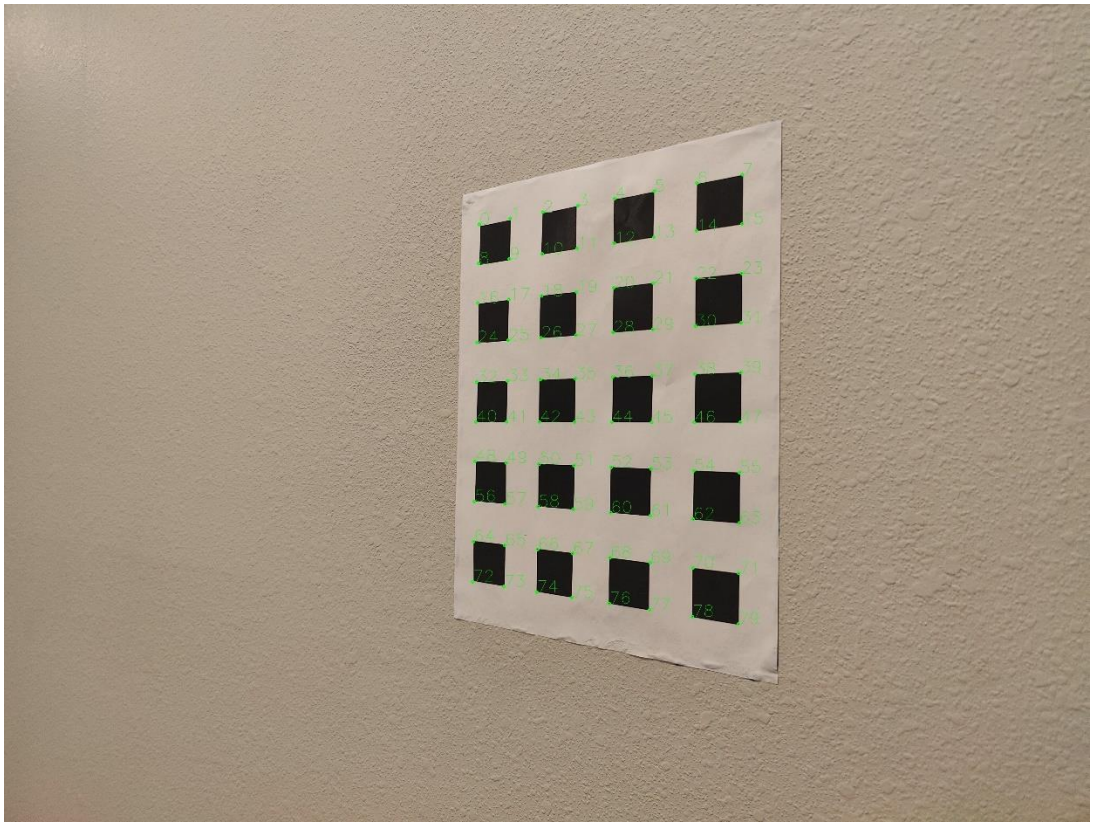
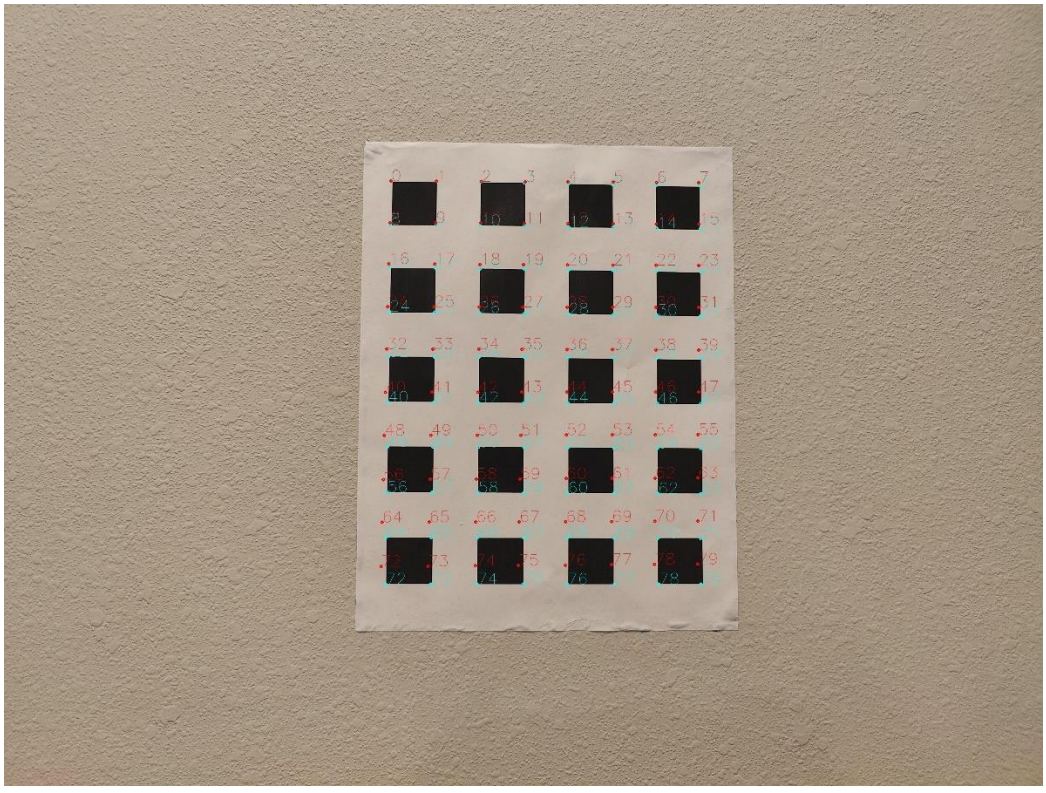


Figure 28 – Reprojected corners of image 15 before(yellow) and after using LM optimization (Difference can be clearly seen in the last row of pattern)



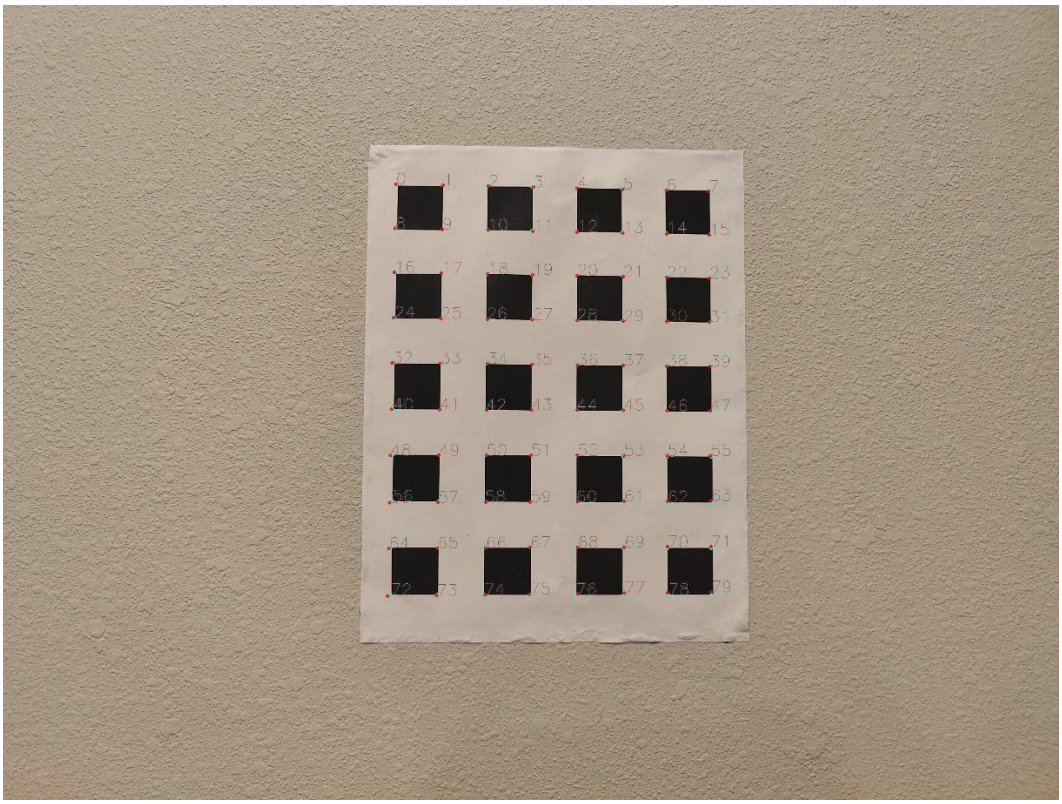
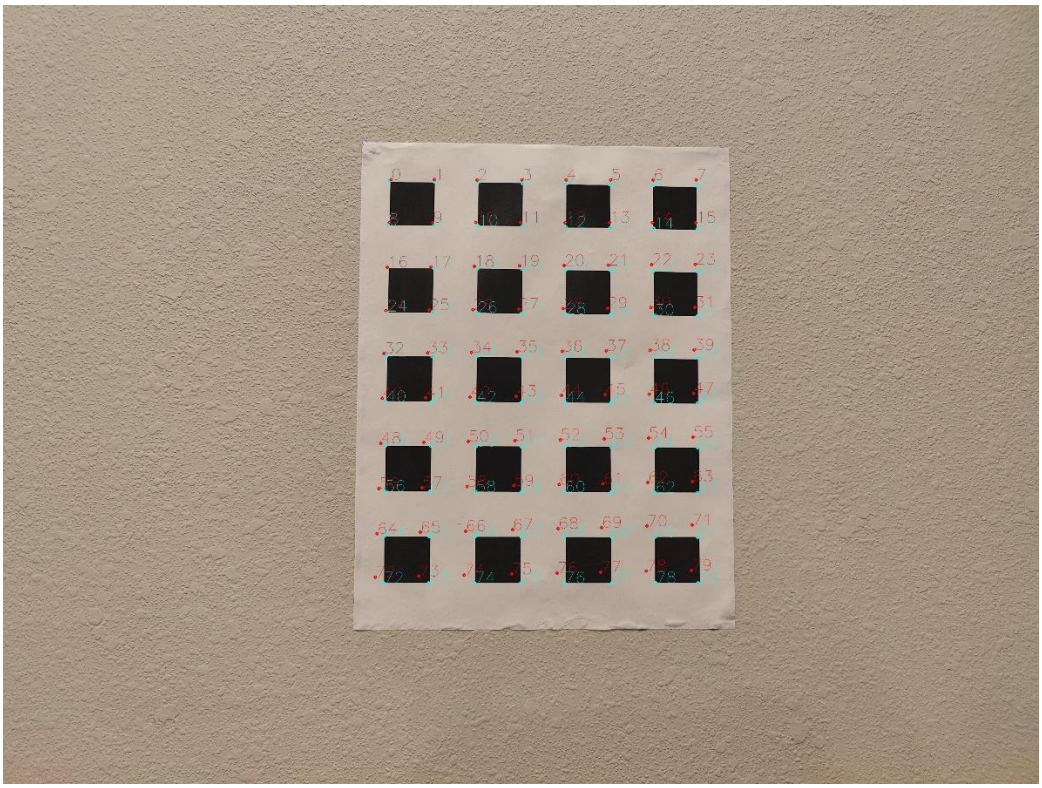


Figure 29 – Corners of image 2 projected on fixed image before and after using LM (Difference of projection can be clearly seen in last row labels)



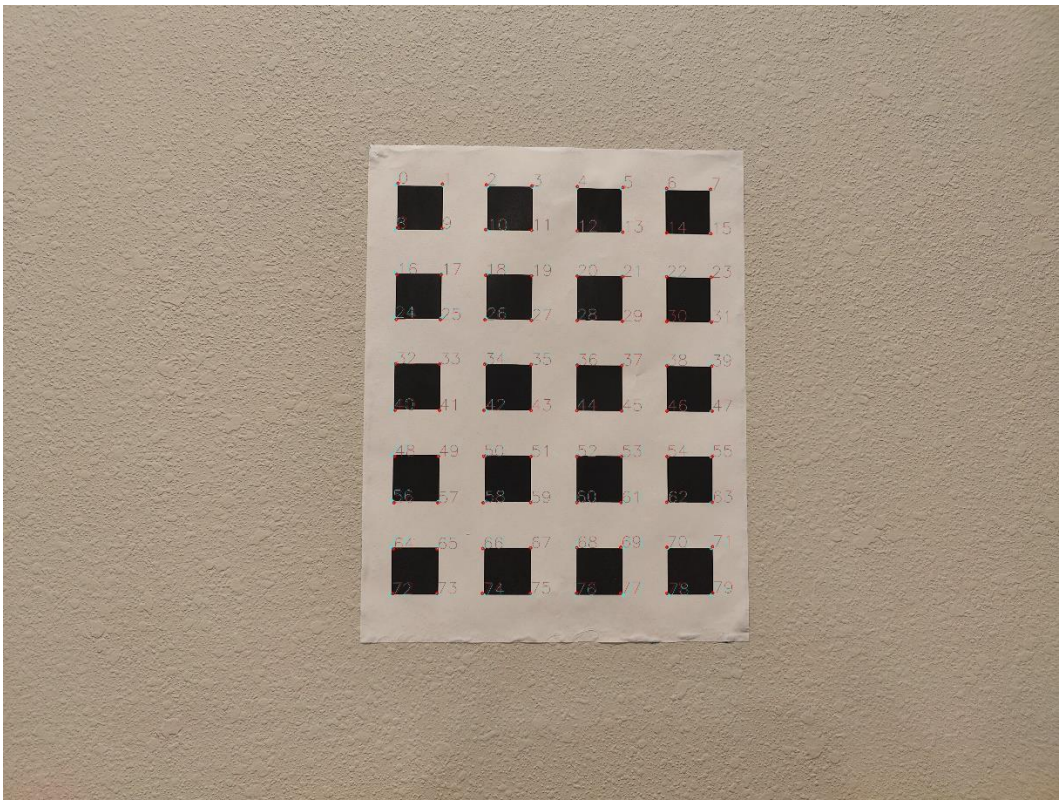
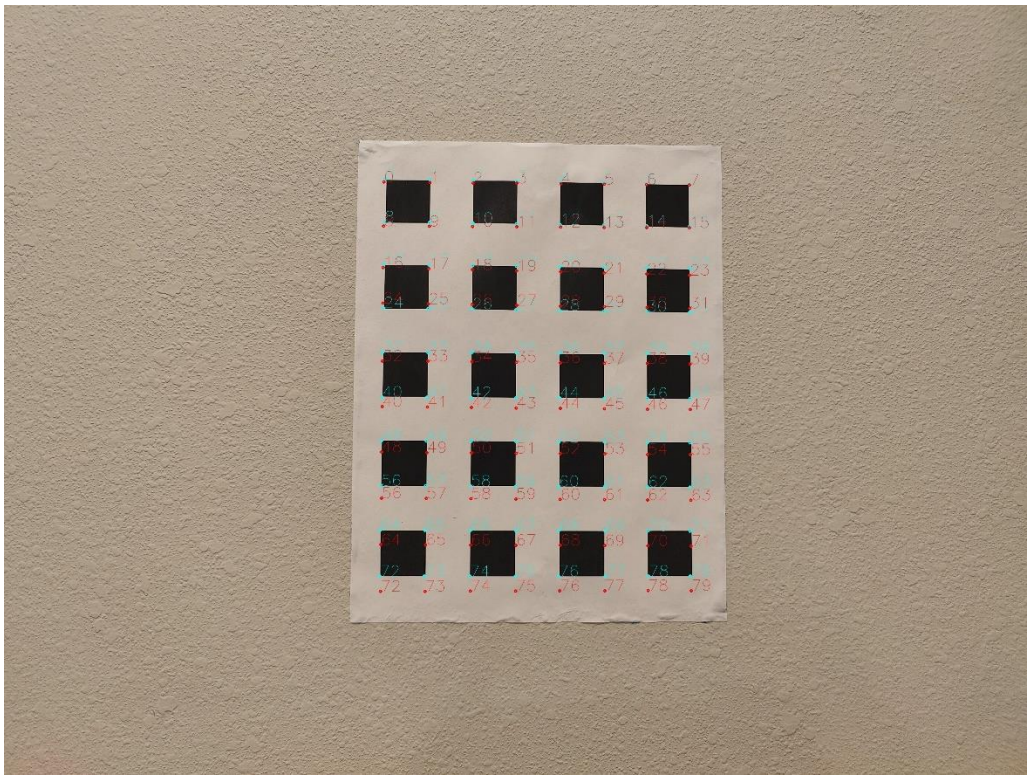


Figure 30 – Corners of image 15 projected on fixed image before(left) and after using LM (Difference of projection can be clearly seen in last row labels)



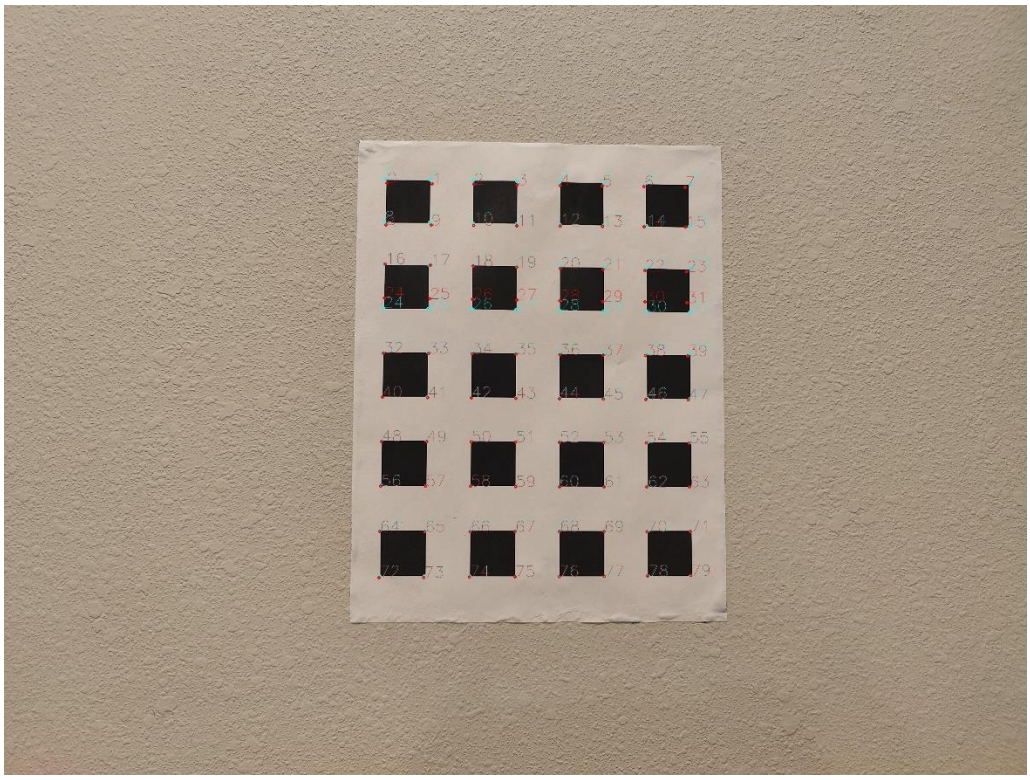
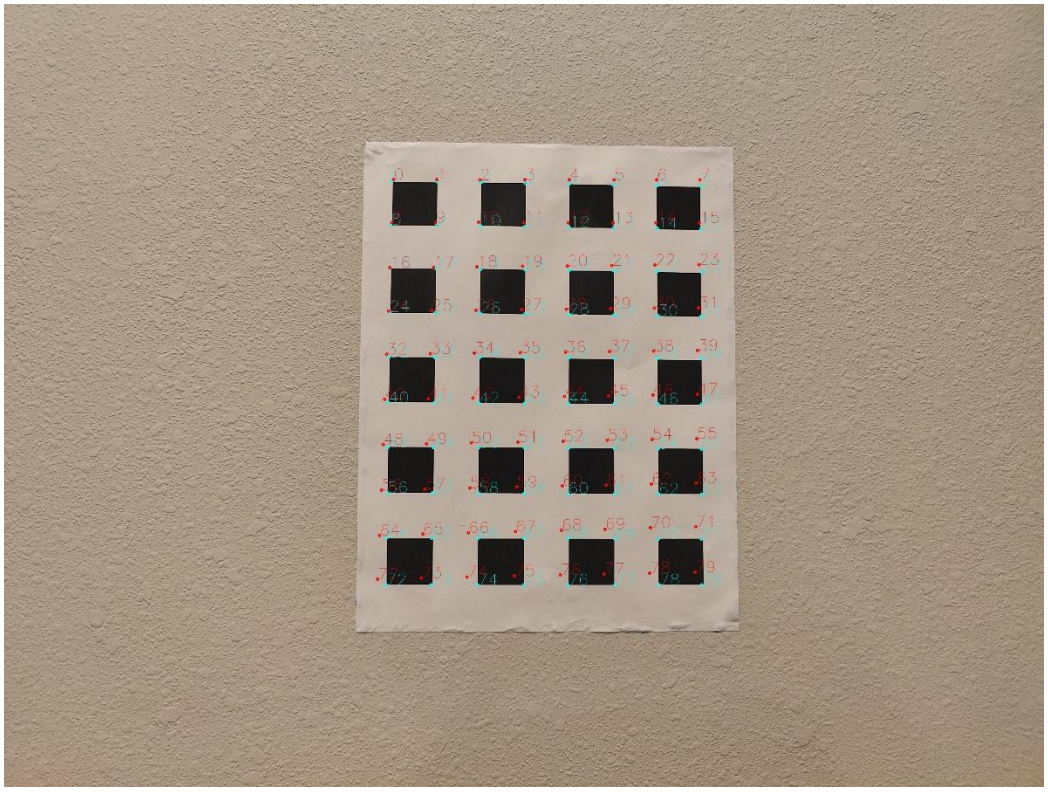


Figure 31 – Corners of image 7 on fixed image before and after radial distortion
(This image is shown to show the considerable improvement in detecting corner by radial despite many mismatching corners)



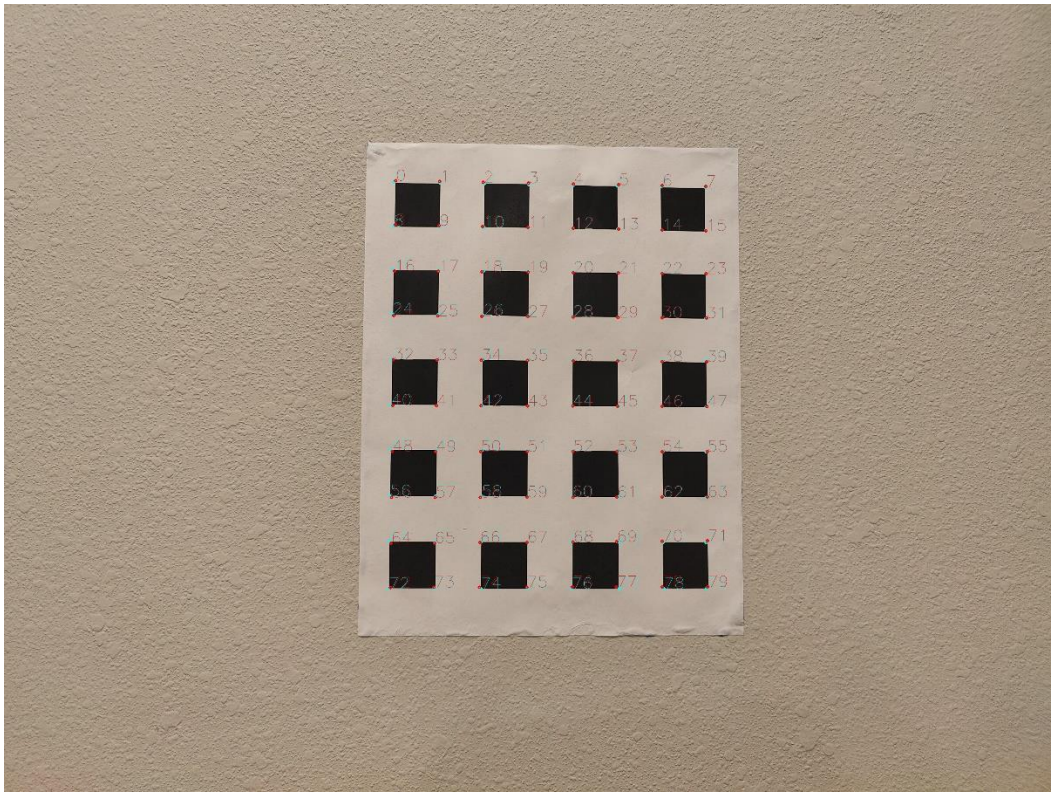


Figure 32 – Corners of image 18 on fixed image before and after radial distortion (Improved projection can be clearly seen in all the labels)

Estimates of K, [R | t] with and without refinement using LM and radial distortion:

Intrinsic camera parameter K:

$$K = \begin{bmatrix} 3357.31 & -34.22 & 1997.4 \\ 0 & 3321.54 & 1466.13 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K\_LMrefined = \begin{bmatrix} 3142.11 & -11.94 & 1874.43 \\ 0 & 3123.66 & 1485.21 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K\_LMrefined\_withRadial = \begin{bmatrix} 3132.68 & -11.77 & 1891.54 \\ 0 & 3114.04 & 1485.63 \\ 0 & 0 & 1 \end{bmatrix}$$

We can observe the improvement in the K matrix as it is refined and incorporated with radial distortion, however the changes from LM to radial are minimal in this case.

Extrinsic camera parameters [R|t]:

For image 2:

$$[\mathrm{R|t}] = \begin{bmatrix} 0.862 & 0.053 & -0.503 & -10.565 \\ 0.005 & 0.993 & 0.114 & -41.867 \\ 0.506 & -0.101 & 0.856 & 182.471 \end{bmatrix}$$

$$[\mathrm{R|t}]\_\mathrm{LMrefined} = \begin{bmatrix} 0.892 & 0.044 & -0.448 & -4.59 \\ -0.0006 & 0.995 & 0.099 & -43.776 \\ 0.450 & -0.088 & 0.888 & 180.47 \end{bmatrix}$$

$$[\mathrm{R|t}]\_\mathrm{LMrefined\_radial} = \begin{bmatrix} 0.895 & 0.045 & -0.443 & -3.653 \\ -0.0009 & 0.995 & 0.100 & -43.76 \\ 0.455 & -0.09 & 0.890 & 180.71 \end{bmatrix}$$

For image 15:

$$[\mathrm{R|t}] = \begin{bmatrix} 0.759 & -0.010 & 0.650 & -16.413 \\ -0.016 & 0.999 & 0.034 & -42.957 \\ -0.650 & -0.037 & 0.758 & 218.482 \end{bmatrix}$$

$$[\mathrm{R|t}]\_\mathrm{LMrefined} = \begin{bmatrix} 0.782 & -0.012 & 0.64 & -9.808 \\ -0.008 & 0.999 & 0.0307 & -46.505 \\ -0.622 & -0.029 & 0.782 & 216.367 \end{bmatrix}$$

$$[\mathrm{R|t}]\_\mathrm{LMrefined\_radial} = \begin{bmatrix} 0.781 & -0.012 & 0.64 & -8.635 \\ -0.008 & 0.999 & -0.0309 & -46.477 \\ -0.626 & -0.029 & 0.780 & 217.056 \end{bmatrix}$$

We can observe that there are some minimal improvements as the metrices are refined with radial distortion after LM refinement.

	Average error	***Variance***	***Max error***
Original parameters	25.629542	2315.491597	457.602187
LM refined only	5.213337	130.435916	114.915329
LM refined with radial distortion	5.191846	130.057710	114.738496

Table 6: The overall projection error of all images with respect to world image. We can observe that the error metrics are reduced with LM and further by radial distortion

	Original parameters	***LM refined only***	***LM refined with radial distortion***
Mean error_Img2	16.039	3.411	3.349
Variance_Img15	85.321	3.316	3.114
Mean error_Img2	36.827	3.1346	3.1339
Variance_Img15	542.213	2.6420	3.2177

Table 7: The projection metrics of image 2 and 15 with world pattern (refers to Figure 27&28). We can observe the mean error and variance are reduced with LM and further by radial distortion.

	Average error	***Variance***	***Max error***
Original parameters	32.911638	3109.513326	501.144430
LM refined only	7.067380	239.373054	142.187948
LM refined with radial distortion	7.067715	227.594958	141.834269

Table 8: The overall projection error of all images with respect to fixed frame. We can observe that the error metrics are reduced with LM and further by radial distortion.

	Original parameters	***LM refined only***	***LM refined with radial distortion***
Mean error_Img2	37.8788	4.9117	4.779
Variance_Img2	465.987	7.097	6.524
Mean error_Img15	44.644	4.3889	4.3686
Variance_Img15	844.680	10.4676	10.8789

Table 9: The projection metrics of image 2 and 15 onto fixed image (refers to Figure 29 and 30). We can observe the mean error and variance are reduced with LM and further by radial distortion.

	Original parameters	*LM refined only*	*LM refined with radial distortion*
Mean error_Img7	17.463	3.1387	3.1362
Variance_Img7	117.493	3.5195	3.5178
Mean error_Img18	10.5498	3.534	3.5246
Variance_Img18	39.922	5.3484	5.3435

Table 10: The projection metrics of image 7 and 18 onto fixed image including radial distortion (refers to Figure 31 and 32). The radial distortion parameters computed are k1 = $2.50306719e^{-9}$ and *k2 =* $3.53273619e^{-15.}$ *We can observe the mean error and variance are reduced with LM and further by radial distortion.*

SOURCE CODE:

```
#!/usr/bin/env python
# coding: utf-8

# <h2><center>ECE661 COMPUTER VISION</center></h2>
# <h3><center>Homework - 8 </center></h3>
# <h3><center>Sahithi Kodali - 34789866</center></h3>
# <h3><center>kodali1@purdue.edu</center></h3>

# In[1]:


# Import libraries needed

import numpy as np
import matplotlib.pyplot as plt
import skimage.io as sk
import cv2
import math
import copy
import pandas as pd
import os
import PIL
get_ipython().run_line_magic('matplotlib', 'inline')
```

```
# In[2]:


#load images from the folder
def load_images_from_folder(path):
    images = []

    #read image
    for img_name in os.listdir(path):
        img = cv2.imread(os.path.join(path + '\\' + img_name))
        images.append(img)

    return images


# In[3]:


#compute the line points from (r,theta) values of each line
def line_points(lines, line_param):

    r = lines[:, 0]
    theta = lines[:, 1]

    a = np.cos(theta)
    b = np.sin(theta)

    x0 = a*r
    y0 = b*r

    p1 = np.array([ x0 + line_param*(-b), y0 + line_param*(a) ]).T
    p2 = np.array([ x0 - line_param*(-b), y0 - line_param*(a) ]).T

    return p1, p2


# In[4]:


#rearrange line points based on the orientation
def rearrange_line_points(pt1, pt2, vertical = True):

    p1 = []
    p2 = []

    for i in range(pt1.shape[0]):
```

```
        if vertical:
            if pt1[i,1] > 0 and pt2[i,1] < 0:
                p1.append(pt1[i])
                p2.append(pt2[i])
            elif pt1[i,1] < 0 and pt2[i,1] > 0:
                p1.append(pt2[i])
                p2.append(pt1[i])
        else:
            if pt1[i,0] > 0 and pt2[i,0] < 0:
                p1.append(pt1[i])
                p2.append(pt2[i])
            elif pt1[i,0] < 0 and pt2[i,0] > 0:
                p1.append(pt2[i])
                p2.append(pt1[i])

    return np.concatenate(p1, axis = 0).reshape((-1,2)), np.concatenate(p2, axis
= 0).reshape((-1,2))


# In[5]:


def distsort_lines(lines, pt1, pt2, ratio, vertical = True ):

    #find the distance threshold using the polar coordinates
    if vertical:
        dist = lines[:,0] * np.cos(lines[:,1])
        dist_abs = np.abs(dist)
        dist_thresh = ratio * (np.max(dist_abs) - np.min(dist_abs))/7
    else:
        dist = lines[:,0] * np.sin(lines[:,1])
        dist_abs = np.abs(dist)
        dist_thresh = ratio * (np.max(dist_abs) - np.min(dist_abs))/9

    sorted_dist_indices = np.argsort(dist, axis = 0)
    dist_sorted = dist[sorted_dist_indices]

    #get the line coords based on the distance threshold
    temp_indices = []
    req_indices = []

    for i in range(dist_sorted.shape[0]-1):

        if i == 0:
            temp_indices.append(sorted_dist_indices[i])
```

```
        if dist_sorted[i+1] - dist_sorted[i] < dist_thresh:
            temp_indices.append(sorted_dist_indices[i+1])
        else:
            req_indices.append(temp_indices)
            temp_indices = [sorted_dist_indices[i+1]]

        if i == dist_sorted.shape[0] - 2:
            req_indices.append(temp_indices)

    #add the average points of points of required indices

    p1 = []
    p2 = []

    for idx in req_indices:
        p1.append(np.average(pt1[idx], axis = 0))
        p2.append(np.average(pt2[idx], axis = 0))

    return np.concatenate(p1, axis = 0).reshape((-1,2)), np.concatenate(p2, axis
= 0).reshape((-1,2))


# In[6]:


#find the intersection points
def intersection_points(vp1, vp2, hp1, hp2):

    #convert point to HC representation
    hp1 = np.append(hp1, np.ones((hp1.shape[0],1)), axis = 1)
    hp2 = np.append(hp2, np.ones((hp2.shape[0],1)), axis = 1)
    vp1 = np.append(vp1, np.ones((vp1.shape[0],1)), axis = 1)
    vp2 = np.append(vp2, np.ones((vp2.shape[0],1)), axis = 1)

    #find lines
    hlines = np.cross(hp1, hp2)
    vlines = np.cross(vp1, vp2)

    #get intersection points of these lines
    inter_points = []

    for i in range(hlines.shape[0]):
        inter = np.cross(vlines, hlines[i])
        inter_points.append(inter[:,:2]/inter[:, 2].reshape((-1,1)))
```

```
    return np.concatenate(inter_points, axis = 0)


# In[7]:


#draw lines using given point coordinates
def draw_lines(img, pt1, pt2, color):

    op_img = img.copy()

    for i in range(pt1.shape[0]):
        x1, y1 = int(pt1[i,0].item()), int(pt1[i,1].item())
        x2, y2 = int(pt2[i,0].item()), int(pt2[i,1].item())

        cv2.line(op_img, (x1, y1), (x2, y2), color, 1)

    return op_img


# In[136]:


# mark a point with its name label for given point coordinates
def mark_points(img, pt, color):

    op_img = img.copy()

    for i in range(pt.shape[0]):
        x, y = int(pt[i,0].item()), int(pt[i,1].item())
        op_img = cv2.circle(op_img, (x,y), 4, color,3)
        op_img = cv2.putText(op_img, str(i), (x,y), cv2.FONT_HERSHEY_SIMPLEX, 2,
color, 1, cv2.LINE_AA)
    return op_img


# In[103]:


def corner_detection(image, i, canny_thresh, theta_param, hough_thresh,
dist_ratio):

    #find the edges using canny operator
    img = np.copy(image)
```

```
    gray_img = cv2.GaussianBlur(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY), (5,5),
1.4) #(3,3) for given data
    canny = cv2.Canny(gray_img, 1.5*canny_thresh, canny_thresh)
    cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/Canny_2/Pic_%d.jpg"%(i),
canny)

    #compute hough transform lines
    hough_lines = cv2.HoughLines(canny, 1, theta_param*np.pi/180, hough_thresh)
    hough_lines = np.squeeze(hough_lines)
    hlp1, hlp2 = line_points(hough_lines, 4000) #1000 for given data
    hough_lines_img = draw_lines(img, hlp1, hlp2, (0,0,255))
    cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/Hough_2/Pic_%d.jpg"%(i),
hough_lines_img)

    #saparate the vertical and horizontal lines based on angle the lines subtend
    #vertical lines
    vert_lines = hough_lines[np.where(np.cos(hough_lines[:,1]) ** 2 > 0.5)]
    vp1, vp2 = line_points(vert_lines, 4000)
    vp1, vp2 = rearrange_line_points(vp1, vp2, vertical = True)
    vp1, vp2 = distsort_lines(vert_lines, vp1, vp2, dist_ratio, vertical = True)
    if vert_lines is not None:
        op_img = draw_lines(img, vp1, vp2, (255,0,0))

    #horizontal lines
    hori_lines = hough_lines[np.where(np.cos(hough_lines[:,1]) ** 2 <= 0.5)]
    hp1, hp2 = line_points(hori_lines, 4000)
    hp1, hp2 = rearrange_line_points(hp1, hp2, vertical = False)
    hp1, hp2 = distsort_lines(hori_lines, hp1, hp2, dist_ratio, vertical = False)
    if hori_lines is not None:
        op_img = draw_lines(op_img, hp1, hp2, (255,0,0))

    cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/HoughRef_2/Pic_%d.jpg"%(i),
op_img)

    #find intersection points of lines
    intersec_points = intersection_points(vp1, vp2, hp1, hp2)
    intersec_img = mark_points(img, intersec_points, (0, 255, 255))

    cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/IntersecPoints_2/Pic_%d.jpg"
%(i), intersec_img)

    return intersec_points


# In[10]:
```

```
def world_coord_grid(size, num_hlines, num_vlines):
    x = np.linspace(0, size*(num_vlines - 1), num_vlines)
    y = np.linspace(0, size*(num_hlines - 1), num_hlines)

    xx, yy = np.meshgrid(x,y)

    grid = np.concatenate([xx.reshape((-1,1)), yy.reshape((-1,1))], axis = 1)

    return grid


# In[11]:


#The Homography matrix is estimated using the world coordinates and the image
coordinates computed using equation AH = B
def compute_H(wc, ic):

    #for each point in world and image, compute A respectively
    def each_A(wc_pt, ic_pt):
        xw, yw = wc_pt[0], wc_pt[1]
        xi, yi = ic_pt[0], ic_pt[1]

        return np.asarray([[xw, yw, 1, 0, 0, 0, -xw * xi, - yw * xi], [ 0, 0, 0,
xw, yw, 1, -xw * yi, - yw * yi]])

    A_all = [each_A(wc[i], ic[i]) for i in range(wc.shape[0])]
    A = np.concatenate(A_all, axis = 0)

    #make a copy of target coordinates and reshape to required dimension
    B = ic.copy()
    B = B.reshape((-1,1))

    #calculate H = inv(A)B
    K = np.dot(np.linalg.pinv(A), B)
    H = np.ones((9,1))
    H[:8, :] = K
    H = H.reshape((3,3))

    return H


# In[12]:
```

```
#compute omega(w) using H matrix and equation Vb = 0
def compute_w(all_H):
    def V_each(H):
        h11, h12, h13 = (H[0,0], H[1,0], H[2,0])
        h21, h22, h23 = (H[0,1], H[1,1], H[2,1])
        return np.asarray([[h11*h21, h11*h22 + h12*h21, h12*h22, h13*h21 +
h11*h23, h13*h22 + h12*h23, h13*h23],
                           [h11**2 - h21**2, 2*h11*h12 - 2*h21*h22, h12**2 -
h22**2, 2*h11*h13 - 2*h21*h23,
                            2*h12*h13 - 2*h22*h23, h13**2 - h23**2]])

    all_V = [V_each(H) for H in all_H]
    V = np.concatenate(all_V, axis=0)
    u,s,v = np.linalg.svd(V)
    b = v[-1]
    w = np.array([[b[0], b[1], b[3]], [b[1], b[2], b[4]], [b[3], b[4], b[5]]])
    return w


# In[13]:


#computer K matrix
def compute_K(W):
    w = np.copy(W)

    v0 = (w[0,1]*w[0,2] - w[0,0]*w[1,2])/(w[0,0]*w[1,1] - w[0,1]**2)
    lamb = w[2,2] - (w[0,2]**2 + v0 * (w[0,1]*w[0,2] - w[0,0]*w[1,2]))/w[0,0]
    alpha_x = np.sqrt(lamb / w[0,0])
    alpha_y = np.sqrt(lamb*w[0,0] / (w[0,0]*w[1,1] - w[0,1]**2))
    s = -(w[0,1] * (alpha_x**2) * alpha_y) / lamb
    u0 = (s*v0/alpha_y) - (w[0,2]*(alpha_x**2)/lamb)

    K = np.array([[alpha_x, s, u0], [0, alpha_y, v0], [0,0,1]])

    return K


# In[14]:


#compute R and t matrix
def compute_Rt(all_H, K):
    all_R = []
    all_t = []
```

```
    for H in all_H:
        r12_t = np.dot(np.linalg.inv(K), H)
        scale = 1/np.linalg.norm(r12_t[:,0])
        r12_t = scale * r12_t
        r3 = np.cross(r12_t[:,0], r12_t[:,1])

        r12t = np.copy(r12_t)
        r12t[:,2] = r3

        u,s,v = np.linalg.svd(r12t)
        R = np.dot(u,v)
        all_R.append(R)
        all_t.append(r12_t[:,2].copy())

    return all_R, all_t


# In[15]:


from scipy.optimize import least_squares


# In[16]:


#change R and t using rodrigeus representation
def modify_params(all_R, all_t, K):
    Rt = []
    for R, t in zip(all_R, all_t):
        phi = np.arccos((np.trace(R) - 1)/2)
        w = phi / (2*np.sin(phi)) * np.asarray([ R[2,1] - R[1,2], R[0,2] -
R[2,0], R[1,0] - R[0,1]])
        Rt.append(np.append(w,t))
    K_vals = np.asarray([K[0,0], K[0,1], K[0,2], K[1,1], K[1,2]])

    ref_params = np.append(K_vals, np.concatenate(Rt))
    return ref_params


# In[17]:


#reconstruct K, R and t matrices
def modify_mat(mod_params):
    n = int((mod_params.shape[0] - 5)/6)
```

```
    k = mod_params[:5]
    K = np.array([[k[0], k[1], k[2]], [0, k[3], k[4]], [0, 0, 1]])

    mod_R = []
    mod_t = []

    for i in range(n):

        w = mod_params[5+i*6 : 8+i*6]
        t = mod_params[8+i*6 : 11+i*6]
        phi = np.linalg.norm(w)
        W = np.array([[0, -w[2], w[1]], [w[2], 0, -w[0]], [-w[1], w[0], 0]])
        R = np.eye(3) + np.sin(phi)/phi * W + (1-np.cos(phi))/(phi**2) *
np.dot(W,W)

        mod_R.append(R)
        mod_t.append(t)

    return mod_R, mod_t, K


# In[18]:


#consider radial distortion
def radial_distortion(coords, k1, k2, x0, y0):
    x = coords[:,0]
    y = coords[:,1]
    r_squ = (x-x0)**2 + (y-y0)**2

    xrad = x + (x-x0) * (k1*r_squ + k2*(r_squ**2))
    yrad = y + (y-y0) * (k1*r_squ + k2*(r_squ**2))

    return np.hstack([xrad.reshape((-1,1)), yrad.reshape(-1,1)])


# In[19]:


#projecting matrix as required
def proj_matrix(H, wc):
    n = wc.shape[0]
    wc_HC = np.concatenate((wc, np.ones((n,1))), axis = 1)

    ic_HC = np.dot(H, wc_HC.T).T
    ic = ic_HC[:, :2]/ic_HC[:, 2].reshape((n,1))
```

```
    return ic


# In[20]:


#cost function to minimize
def LM_cost_func(ref_params, intersec_coords, wc_coords, rad_dist = False):

    if rad_dist:
        R_all, t_all, K = modify_mat(ref_params[:-2])
        k1 = ref_params[-2]
        k2 = ref_params[-1]
        print(k1)
        x0 = ref_params[2]
        y0 = ref_params[4]
    else:
        R_all, t_all, K = modify_mat(ref_params)

    projections = []

    for R,t in zip(R_all, t_all):
        Rt = np.concatenate([R[:, 0:1], R[:, 1:2], t.reshape((-1,1))], axis=1)
        H = np.dot(K, Rt)
        proj = proj_matrix(H, wc_coords)

        if rad_dist:
            proj = radial_distortion(proj, k1, k2, x0, y0)

        projections.append(proj)

    proj_coords = np.concatenate(projections, axis = 0)
    img_coords = np.concatenate(intersec_coords, axis = 0)

    diff = proj_coords - img_coords

    return diff.flatten()


# In[148]:


#projecting image views to world coords
def project_wc_metrics(wc_coord, inter_coord, valid_img_ids, imgs, mod_params,
refine = False):
```

```
    if mod_params.shape[0] % 6 == 1:
        R_all, t_all, K = modify_mat(mod_params[:-2])
        k1 = mod_params[-2]
        k2 = mod_params[-1]
        x0 = mod_params[2]
        y0 = mod_params[4]
    else:
        R_all, t_all, K = modify_mat(mod_params)

    H_all = []
    diff_all =[]

    for R,t in zip(R_all, t_all):
        Rt = np.concatenate([R[:, 0:1], R[:, 1:2], t.reshape((-1,1))], axis=1)
        H = np.dot(K, Rt)
        H_all.append(H)

    for i, H in enumerate(H_all):
        img_id = valid_img_ids[i]
        valid_img = imgs[img_id]

        ic_proj = proj_matrix(H, wc_coord)

        if mod_params.shape[0] % 6 == 1:
            ic_proj = proj_matrix(H, wc_coord)
            ic_proj = radial_distortion(ic_proj, k1, k2, x0, y0)
        else :
            ic_proj = proj_matrix(H , wc_coord )

        proj_img = mark_points(valid_img, ic_proj, (0,255,255))
        cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/ProjImg_2/Pic_%d.jpg"%(i
mg_id), proj_img)
#         cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/ProjImg_2_lm/Pic_%d.jp
g"%(img_id), proj_img)
#         cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/ProjImg_2_rad/Pic_%d.j
pg"%(img_id), proj_img)


        diff = inter_coord[i] - ic_proj
        diff_all.append(diff)

    all_metrics = overall_metrics(np.array(diff_all).flatten())
    each_metrics = calc_metrics(np.array(diff_all).flatten())

    return all_metrics, each_metrics
```

```
# In[83]:


#compute metrics overall
def overall_metrics(diff):
    diff = diff.reshape((-1,2))
    d_norm = np.linalg.norm(diff, axis = 1)
    avg_d = np.average(d_norm)
    var_d = np.var(d_norm)
    max_d = np.max(d_norm)

    return np.array([avg_d, var_d, max_d])


# In[84]:


#compute metrics with respect to each image
def calc_metrics(diff):
    diff = diff.reshape((-1,2))
    d_norm = np.linalg.norm(diff, axis = 1)
    n = int(d_norm.shape[0]/80)

    metrics = []

    for i in range(n):
        metrics_img = {}
        d = d_norm[i*80 : i*80 + 80]

        metrics_img['Mean :'] = np.average(d)
        metrics_img['Variance :'] = np.var(d)
        metrics_img['Max_distance :'] = np.max(d)

        metrics.append(metrics_img)

    return metrics


# In[142]:


#reproject views to a fixed image coords
def project_fixed_metrics(fixed_img_coord, inter_coord, valid_img_ids, imgs,
mod_params, refine = False):
```

```
    fix_img_id = valid_img_ids[fixed_img_coord]
    fix_img = np.copy(imgs[fix_img_id])
    fix_img_inter = inter_coord[fix_img_id]
    fix_img = mark_points(fix_img, fix_img_inter, (255,255,0))

    if refine:
        R_all, t_all, K = modify_mat(mod_params[:-2])
        k1 = mod_params[-2]
        k2 = mod_params[-1]
        x0 = mod_params[2]
        y0 = mod_params[4]
    else:
        R_all, t_all, K = modify_mat(mod_params)

    H_all = []
    diff_all =[]

    for R,t in zip(R_all, t_all):
        Rt = np.concatenate([R[:, 0:1], R[:, 1:2], t.reshape((-1,1))], axis=1)
        H = np.dot(K, Rt)
        H_all.append(H)

    fix_img_H = H_all[fix_img_id]

    for i, H in enumerate(H_all):

        if i == fix_img_id:
            continue

        img_id = valid_img_ids[i]
        H_img = np.dot(fix_img_H, np.linalg.inv(H))
        fixproj = proj_matrix(H_img, inter_coord[i])
        fixproj_img = mark_points(fix_img, fixproj, (0,0,255))
        cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/ReprojImg_2/Pic_%d.jpg"%
(img_id), fixproj_img)
#         cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/ReprojImg_2_lm/Pic_%d.
jpg"%(img_id), fixproj_img)
#         cv2.imwrite("D:/Purdue/ECE661_CV/HW8/HW8_outputs/ReprojImg_2_rad/Pic_%d
.jpg"%(img_id), fixproj_img)


        diff = fix_img_inter - fixproj
        diff_all.append(diff)
```

```
    all_metrics = overall_metrics(np.array(diff_all).flatten())
    each_metrics = calc_metrics(np.array(diff_all).flatten())

    return all_metrics, each_metrics


# ### Given dataset

# In[24]:


#data paths
folder_path = "D:\Purdue\ECE661_CV\HW8\HW8-Files"
data_path = os.path.join( folder_path + "\\Dataset1")


# In[25]:


#load images and find the intersections
data1 = load_images_from_folder(data_path)
wc_coords = world_coord_grid(10, 10, 8)

all_imgs_H = []
all_imgs_inter = []
all_validimgs_id = []
for i, img in enumerate(data1):
    inter_coords = corner_detection(img, i, 255, 0.5, 50, 0.25)

    if inter_coords.shape[0] == 80:
        H = compute_H(wc_coords, inter_coords)
        all_imgs_H.append(H)
        all_imgs_inter.append(inter_coords)
        all_validimgs_id.append(i)

print(len(all_validimgs_id))
print(all_validimgs_id)


# In[30]:


#compute K,R,t
w = compute_w(all_imgs_H)
K = compute_K(w)
all_R, all_t = compute_Rt(all_imgs_H, K)
```

```
# In[31]:


#parameters before and after LM
modified_params = modify_params(all_R, all_t, K)
res = least_squares(LM_cost_func, modified_params, method = 'lm', args =
[all_imgs_inter, wc_coords])
R_ref, t_ref, K_ref = modify_mat(res.x)


# In[32]:


#parameters before and after radial
res_rad = least_squares(LM_cost_func, np.append(modified_params,
np.array([0,0])), method = 'lm', args = [all_imgs_inter, wc_coords, True ])
R_ref_rad, t_ref_rad, K_ref_rad = modify_mat(res_rad.x)


# In[33]:


#K matrix
print(K)
print(K_ref)
print(K_ref_rad)


# In[73]:


#R and t matrices
for i in [3 , 15, 21 , 35]:
    print(" Projecting the world coordinates to image -", all_validimgs_id[i])
    print(" R and t without lm :\n",all_R[i] , all_t[i])
    print(" R and t with lm :\n",R_ref[i] , t_ref[i])
    print(" R and t with lm and radial :\n",R_ref_rad[i] , t_ref_rad[i])
    print('')


# In[183]:


#overall metrics with and without LM and radial
print ("k1 , k2:", res_rad.x[-2:])
```

```
print("Without lm:", overall_metrics(LM_cost_func(modified_params,
all_imgs_inter, wc_coords)))
print("With lm:", overall_metrics(LM_cost_func(res.x, all_imgs_inter,
wc_coords)))
print("With lm and radial:", overall_metrics(LM_cost_func(res_rad.x,
all_imgs_inter, wc_coords, rad_dist = True)))


# In[86]:


#without lm
all_metr, each_metr = project_wc_metrics(wc_coords, all_imgs_inter,
all_validimgs_id, data1, modified_params, False)


# In[88]:


#with lm
all_metr_lm, each_metr_lm = project_wc_metrics(wc_coords, all_imgs_inter,
all_validimgs_id, data1, res.x, False)


# In[90]:


#with rad
all_metr_lm_rad, each_metr_lm_rad = project_wc_metrics(wc_coords, all_imgs_inter,
all_validimgs_id, data1, res_rad.x, True)


# In[173]:


print(all_metr , all_metr_lm, all_metr_lm_rad)


# In[172]:


#image - 16
print(all_validimgs_id[15], each_metr[15] , each_metr_lm[15],
each_metr_lm_rad[15])

#image - 37
```

```
print(all_validimgs_id[35], each_metr[35] , each_metr_lm[35],
each_metr_lm_rad[35])


# In[94]:


#without lm
all_metr_fixproj, each_metr_fixproj = project_fixed_metrics(1, all_imgs_inter,
all_validimgs_id, data1, modified_params, False)


# In[96]:


#with lm
all_metr_fixproj_lm, each_metr_fixproj_lm = project_fixed_metrics(1,
all_imgs_inter, all_validimgs_id, data1, res.x, False)


# In[101]:


#with rad
all_metr_fixproj_lm_rad, each_metr_fixproj_lm_rad = project_fixed_metrics(0,
all_imgs_inter, all_validimgs_id, data1, res_rad.x, True)


# In[99]:


print(all_metr_fixproj , all_metr_fixproj_lm_rad, all_metr_fixproj_lm)


# In[175]:


for i in [15 , 21 , 33, 35 ]:
    print (" image :", all_validimgs_id[i])
    print (" Without lm :", each_metr_fixproj[i])
    print (" With lm and radial :", each_metr_fixproj_lm[i])
    print (" With lm and radial :", each_metr_fixproj_lm_rad[i])


# ### Own dataset

# In[110]:
```

```
odata_path = os.path.join( folder_path + "\\Dataset2")
data2 = load_images_from_folder(odata_path)
plt.imshow(data2[0])


# In[150]:


#load images and find the intersections
wc_coords2 = world_coord_grid(10, 10, 8)

all_imgs_H2 = []
all_imgs_inter2 = []
all_validimgs_id2 = []
for i, img in enumerate(data2):
    inter_coords = corner_detection(img, i, 255, 0.5, 140, 0.3)

    if inter_coords.shape[0] == 80:
        H = compute_H(wc_coords, inter_coords)
        all_imgs_H2.append(H)
        all_imgs_inter2.append(inter_coords)
        all_validimgs_id2.append(i)

print(len(all_validimgs_id2))
print(all_validimgs_id2)


# In[113]:


#compute K,R,t
w2 = compute_w(all_imgs_H2)
K2 = compute_K(w2)
all_R2, all_t2 = compute_Rt(all_imgs_H2, K2)


# In[114]:


#parameters before and after LM
modified_params2 = modify_params(all_R2, all_t2, K2)
res2 = least_squares(LM_cost_func, modified_params2, method = 'lm', args =
[all_imgs_inter2, wc_coords2])
R_ref2, t_ref2, K_ref2 = modify_mat(res2.x)
```

```
# In[115]:


#parameters before and after radial
res_rad2 = least_squares(LM_cost_func, np.append(modified_params2,
np.array([0,0])), method = 'lm', args = [all_imgs_inter2, wc_coords2, True ])
R_ref_rad2, t_ref_rad2, K_ref_rad2 = modify_mat(res_rad2.x)


# In[116]:


#K matrix
# np.set_printoptions(suppress=True, formatter={'float_kind':'{:f}'.format})
print(K2)
print(K_ref2)
print(K_ref_rad2)


# In[182]:


#R and t matrices
for i in [2 , 15, 12 , 18]:
    print(" Projecting the world coordinates to image -", all_validimgs_id2[i])
    print(" R and t without lm :\n",all_R2[i] , all_t2[i])
    print(" R and t with lm :\n",R_ref2[i] , t_ref2[i])
    print(" R and t with lm and radial :\n",R_ref_rad2[i] , t_ref_rad2[i])
    print('')


# In[149]:


all_metr2, each_metr2 = project_wc_metrics(wc_coords2, all_imgs_inter2,
all_validimgs_id2, data2, modified_params2, False)


# In[147]:


all_metr_lm2, each_metr_lm2 = project_wc_metrics(wc_coords2, all_imgs_inter2,
all_validimgs_id2, data2, res2.x, True)
```

```
# In[145]:


all_metr_lm_rad2, each_metr_lm_rad2 = project_wc_metrics(wc_coords2,
all_imgs_inter2, all_validimgs_id2, data2, res_rad2.x, True)


# In[124]:


print(overall_metrics(LM_cost_func(modified_params2, all_imgs_inter2,
wc_coords2)))
print(overall_metrics(LM_cost_func(res2.x, all_imgs_inter2, wc_coords2)))
print(overall_metrics(LM_cost_func(res_rad2.x, all_imgs_inter2, wc_coords2,
rad_dist = True)))


# In[181]:


#image - 2
print(all_validimgs_id2[2], each_metr2[2] , each_metr_lm2[2],
each_metr_lm_rad2[2])

#image - 15
print(all_validimgs_id2[15], each_metr2[15] , each_metr_lm2[15],
each_metr_lm_rad2[15])


# In[143]:


all_metr_fixproj2, each_metr_fixproj2 = project_fixed_metrics(0, all_imgs_inter2,
all_validimgs_id2, data2, modified_params2, False)


# In[139]:


all_metr_fixproj_lm2, each_metr_fixproj_lm2 = project_fixed_metrics(0,
all_imgs_inter2, all_validimgs_id2, data2, res2.x, True)


# In[137]:
```

```
all_metr_fixproj_lm_rad2, each_metr_fixproj_lm_rad2 = project_fixed_metrics(0,
all_imgs_inter2, all_validimgs_id2, data2, res_rad2.x, True)


# In[132]:


print(all_metr_fixproj2 , all_metr_fixproj_lm2, all_metr_fixproj_lm_rad2)


# In[180]:


for i in [2, 7, 15, 18]:
    print (" image :", all_validimgs_id2[i])
    print (" Without lm :", each_metr_fixproj2[i])
    print (" With lm :", each_metr_fixproj_lm2[i])
    print (" With lm and radial :", each_metr_fixproj_lm_rad2[i])
```

*****