Assignment 9: ECE 661

Dwarakanath: djampani@purdue.du

November 22, 2016

1 Introduction

The Goal of this Assignment is to implement Zhang's Algorithm to estimate the extrinsic and intrinsic parameters of camera. The code is written in python with own implementation of Zhang's Algorithm and used opencv for Hough line fitting, Canny Edge detection and Levenberg Marquadt.

2 Corner Detection

- Apply opencv Canny edge detector to the image.
- Apply opencv Hough fit line method to find the lines in the image.
- The rho value for detecting hough lines is 1 for opencv Houghline.
- The theta is angle in radians and passed a value of $\pi/180$
- Hough line gives many lines , so when finding the corners we can do non maximal supression.
- we can sort the lines by angles and group them to vertical and horizontal line. vertical line if the angle is +-20 else it is horizontal line.
- for multiple lines we can pick first line (this works fine too) because used sub pixel accuracy and also LM optimization and the best way is find the line that gets maximum number of pixels and discard rest of the lines.
- The intersection of lines gives the corners.
- The corners are labelled from left to right and top to bottom for all images.

3 Camera Calibration

3.1 calculating intrinsic parameters

Camera calibration is based on Zhang's Algorithm. The algorithm estimates the intrinsic parameters of a camera which is represented by and given by 3*3 matrix

and extrinsic parameters R and t which are 3*3 and 3*1 respectively. The algorithm estimates the parameters by assuming the pattern is in Z=0 plane. The homogeneous representation of the pixel coordinates is given by $[x, y, w]^T$, so we can write $\vec{x} = k[R|\vec{t}][x, y, 0, w]^T = H\vec{x_w}$. we can write $\vec{x_m} = [X, Y, W]^T$ and $H = [\vec{h1}, \vec{h2}, \vec{h3}]$. so we can estimate R and $\vec{t}$ by estimating H. The Zhangs algorithm is based on the camera image of absolute conic Ω_∞ and given by $\omega = K^{-T}k^{-1}$. we know that plane intersects conic at two points and they obey the equation $x^T wx = 0$. so we get equations $\vec{h1}^T \omega \vec{h1} = \vec{h2}^T \omega \vec{h}262$ and $\vec{h1}^T \omega \vec{h2} = 0$ we can convert equations to the form $\begin{bmatrix} h11 & h12 & h13 \end{bmatrix} \begin{bmatrix} w11 & w12 & w13 \\ w21 & w22 & w23 \\ w31 & w32 & w33 \end{bmatrix} \begin{bmatrix} h21 \\ h22 \\ h23 \end{bmatrix} = 0$ which implies $\vec{v_{12}}^T \vec{b} = 0$ and same we can write $\vec{h_1}^T \omega \vec{h_1} - \vec{h_2}^T \omega \vec{h_2} = 0$ and expressed as $(\vec{v_{11}} - \vec{v_{22}})\vec{b} = 0$ and we can stack these equations as $v = [\vec{v_{12}}^T, (\vec{v_{12}} - \vec{v_{22}})^T]^T$ and v is a 2*6 matrix and $v_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{j1}h_{i2} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{j3}h_{i1} \\ h_{i3}h_{j2} + h_{j3}h_{i2} \\ h_{i3}h_{j3} \end{bmatrix}$. we can use min 3 camera positions and stack them in a matrix and solve using linear least square minimization for b, but we use many more positions to get good accuracy. after getting ω , make $\omega = K^{-T}K^{-1}$.
K= $\begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$
The terms used in this assignment are defined here(for more clarity).
P=projection matrix
ω = The image of absolute conic.
K=camera intrinsic parameters.
H=Homography between the image and world coordinates.
$\vec{I}, \vec{J}$ =circular points
Ω_∞=absolute conic.
R= Rotation matrix
$\vec{t}$=translation vector.

3.2 calculating the extrinsic parameters

Extrinsic parameters are R and $\vec{t}$we can calculate parameters by using $k^{-1}[\vec{h_1}\vec{h_2}\vec{h_3}] = [\vec{r_1}\vec{r_2}\vec{r_3}]$and $\vec{r_1} = k^{-1}\vec{h1}$ and $\vec{r2} = k^{-1}\vec{h_2}$ and $\vec{r_3} = \vec{r_1} \times \vec{r_2}$ and $\vec{t} = k^{-1}\vec{h3}$ To make rows ad columns of R orthonormal w can condition R by doing SVD and SVD(R)=UDV^T and making $R = UV^T$.

3.3 refining the parameters

Notation:
$\vec{x_{m,j}}$: The jth salient point on the calibration pattern.
$\vec{x_{i,j}}$: The actual image point for $\vec{x_{m,j}}$ in the i th camera position.
$\vec{\bar{x}_{i,j}}$: The projected image point for $\vec{x_{m,j}}$ using P for i th position.

R_i : The Rotation matrix
$\vec{t_i}$: The translation vector for i th position.
k: The camera calibration matrix.
The parameters estimated does not give intuitive sense how good they are . To improve the accuracy of the parameters we use Levenberg Marquardt Optimization. if we have calculated the parameters correctly we have$\vec{x_{i,j}} = \vec{\hat{x}_{i,j}}$ we can calculate the euclidean distance between two points that gives how far the calibration with respect to the j th salient point in the i th position in the camera. we can sum the distance for all points in all camera positions. we can write $d^2_{geom} = ||\vec{X} - f(\vec{p})||^2$. we can minimize the parameters using LM method , a mere extension for Gradient descent optimization.

3.4 Incorporating the radial distortion

when a camera has short focal length it exhibits radial distortion. we can model the radial distortion as follows. let $(\hat{x}, \hat{y})$ be the predicted position of a pixel using the pin hole camera. $\hat{x_{rad}} = \hat{x} + (\hat{x} - x_0)[k_1 r^2 + k_2 r^4]$
$\hat{y_{rad}} = \hat{y} + (\hat{y} - y_0)[k_1 r^2 + k_2 r^4]$
$r^2 = (\hat{x} - x_0)^2 + (\hat{y} - y_0)^2$
we can invoke the non linear least square minimization to estimate the parameters R,t and K. we can also minimize the parameters by optimizing the LM method.

4 Results

The corners are numbered from left to right and top to bottom .I have marked on all images, in case i missed, The ordering is from left to right and right and top to bottom.All corners in the world coordinate are assumed to be separated by 1 inch. The images and results are displayed in the following order

- first Hough lines,edges,corners (for 2 images)
- Reprojection for 3 images
- before reprojection for 3 images.
- before LM optimization for 2 images
- after LM optimization for 2 images.
- Internal Parameters
- Each image(this image is from the dataset , does not have any corners, just to show how the image looks for the parameters)

- External parameters before refined and after for the corresponding image above
- Table containing mean error and variance error for before optimization and after optimization.
- Results repeated for my own dataset.
- for extra credit computed radial distortion parameters for two datasets
- quantitative results for radial distortion parameters.

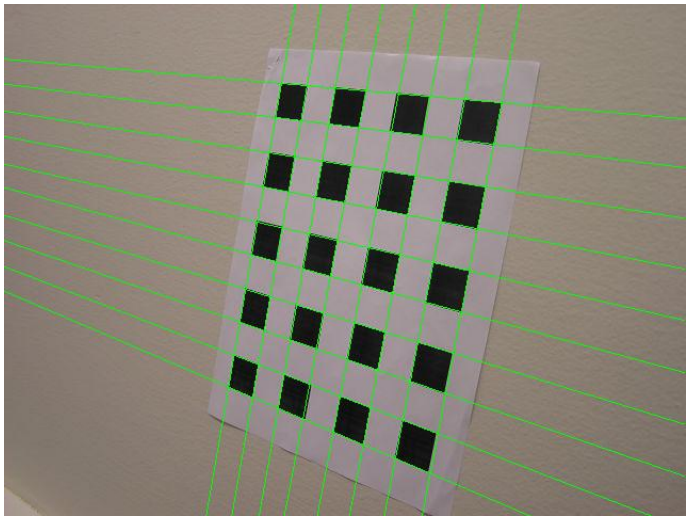

Figure 1: Hough lines

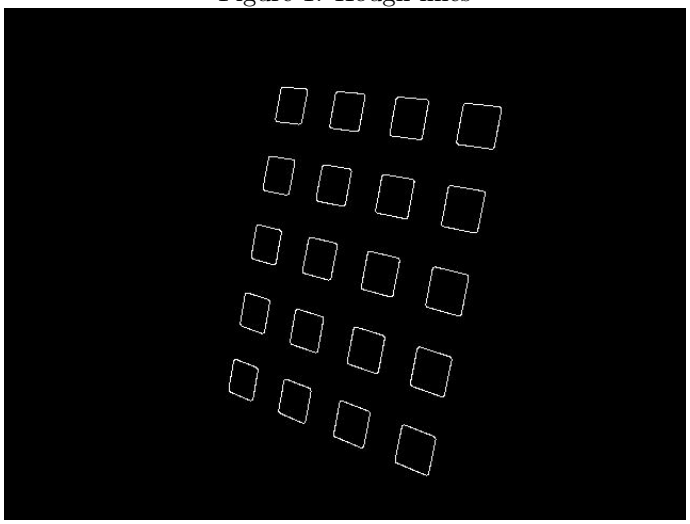

Figure 2: Edges.

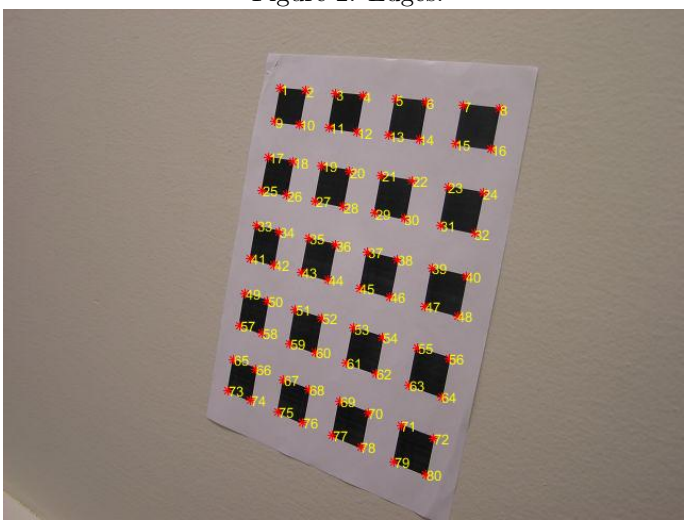

Figure 3: corners.

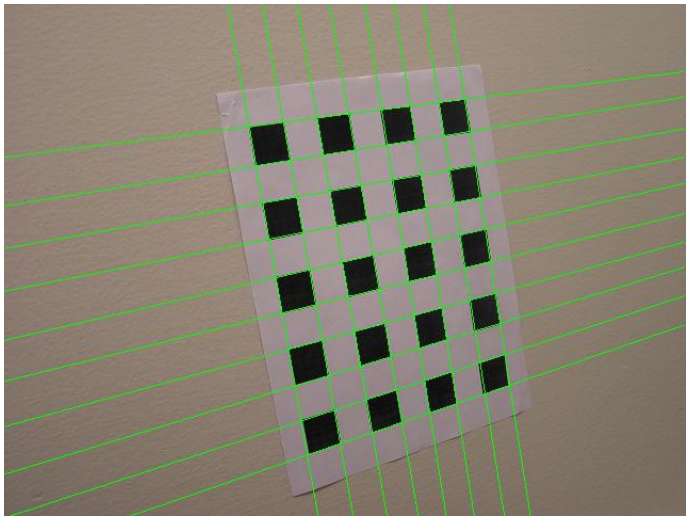

Figure 4: Hough lines

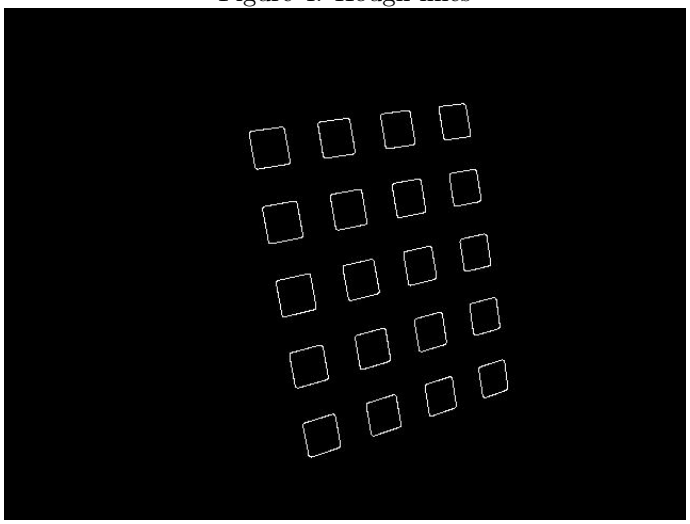

Figure 5: Edges.

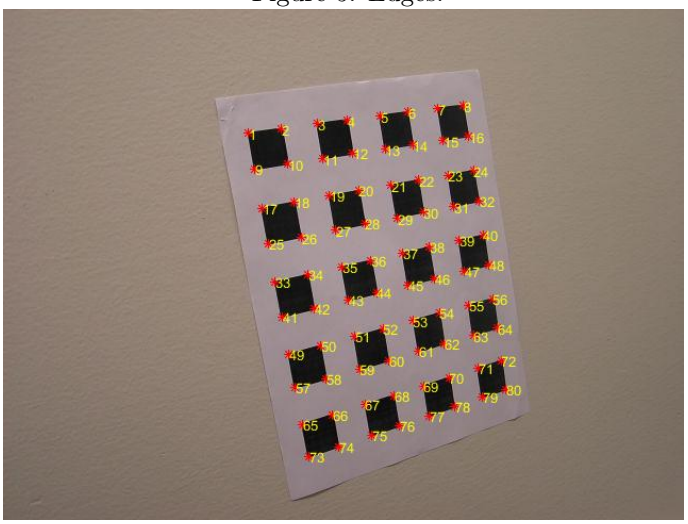

Figure 6: corners.

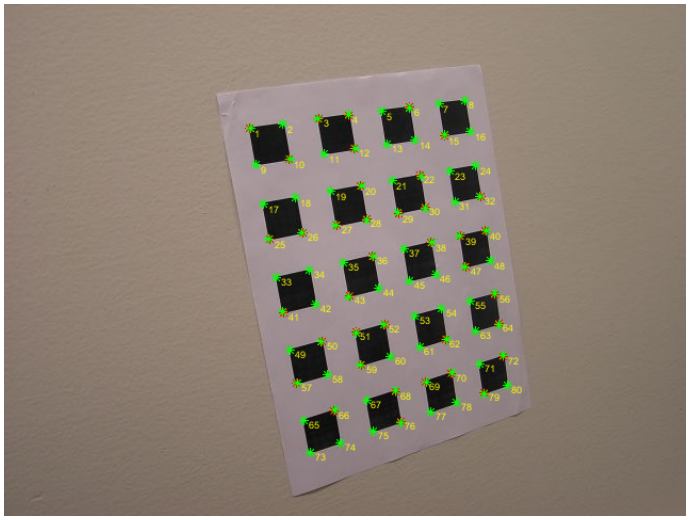

Figure 7: Reprojection with labels

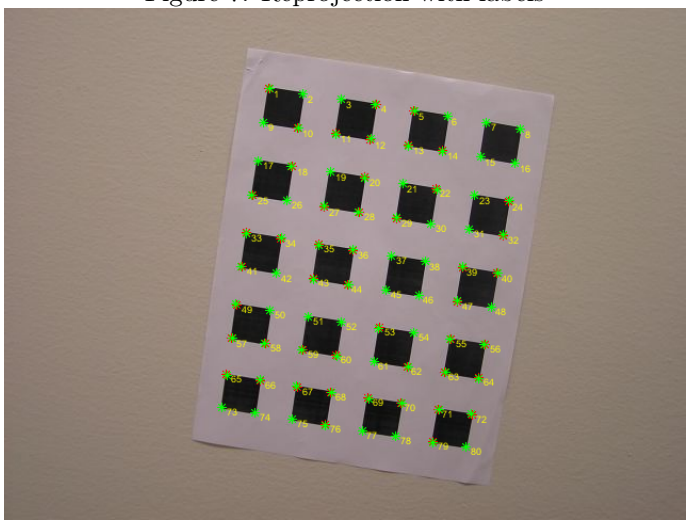

Figure 8: Reprojection with labels.

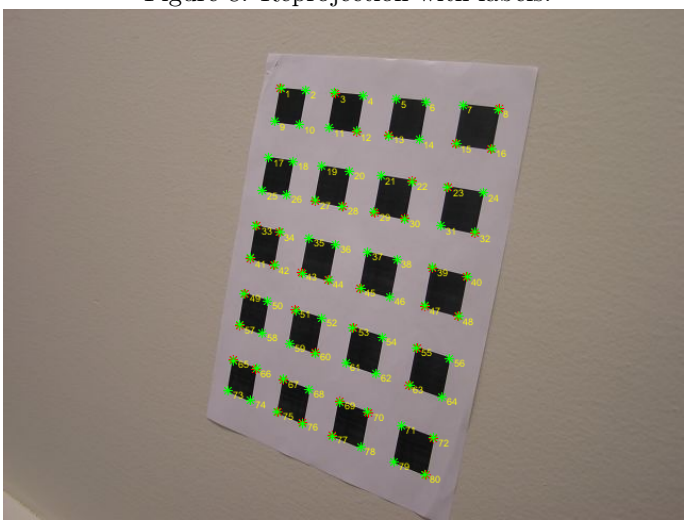

Figure 9: Reprojection with labels

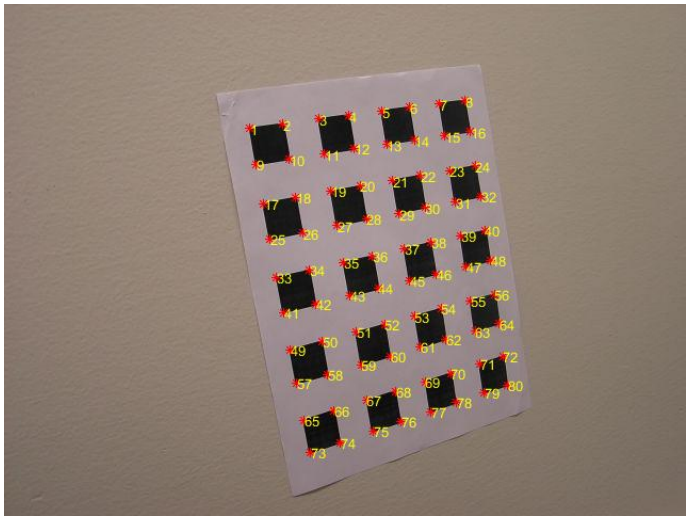

Figure 10: before Reprojection

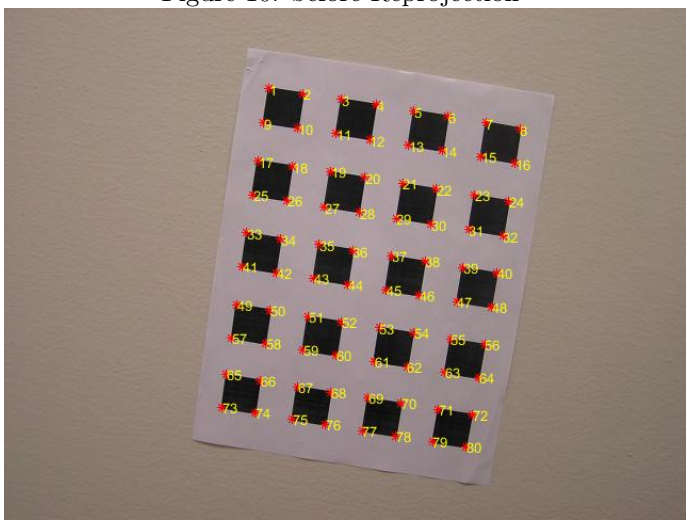

Figure 11: before Reprojection .

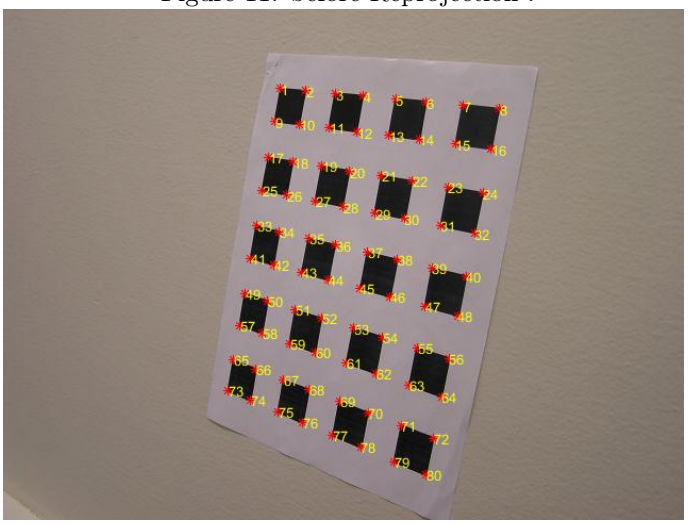

Figure 12: before Reprojection

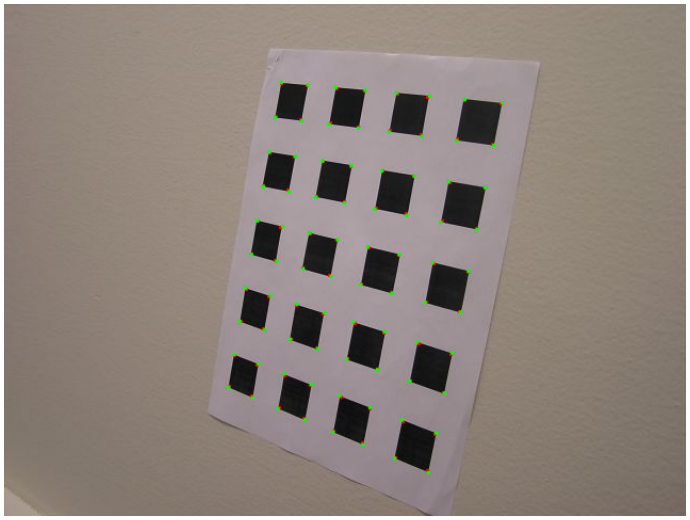

Figure 13: before LM optimization

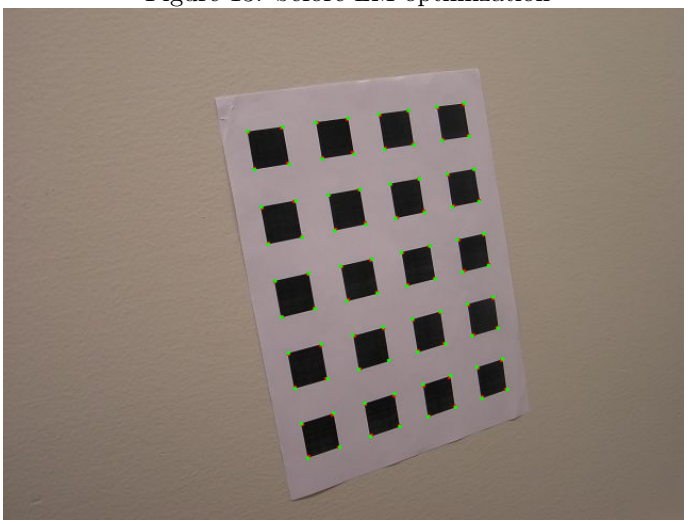

Figure 14: before LM optimization

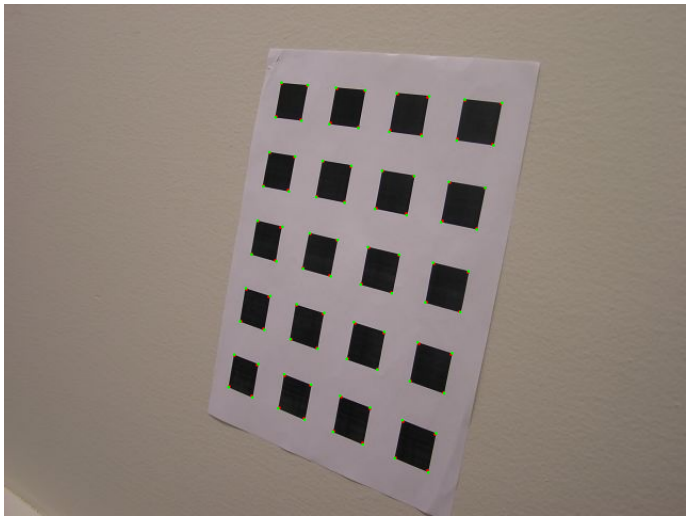

Figure 15: After LM optimization

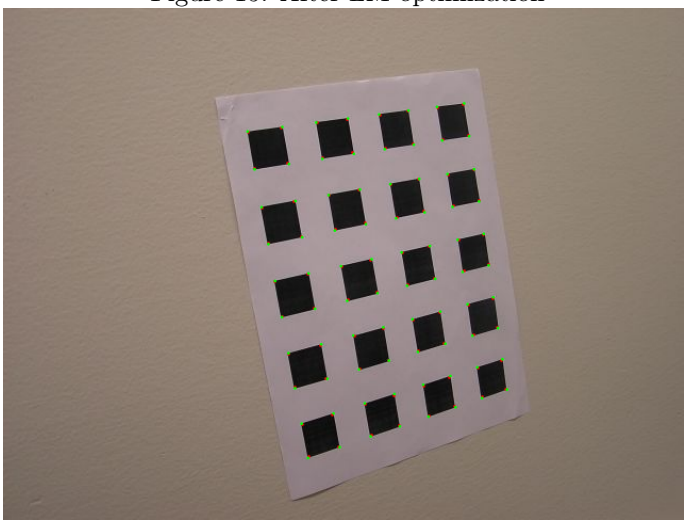

Figure 16: After LM optimization

The internal parameters for the 40 image dataset is $\begin{bmatrix} 717.69188198 & 1.71980609 & 318.89659066 \\ 0. & 718.72320119 & 236.95118643 \\ 0. & 0. & 1. \end{bmatrix}$

The external parameters for Images are below and multiply each element in R by 10^-5.

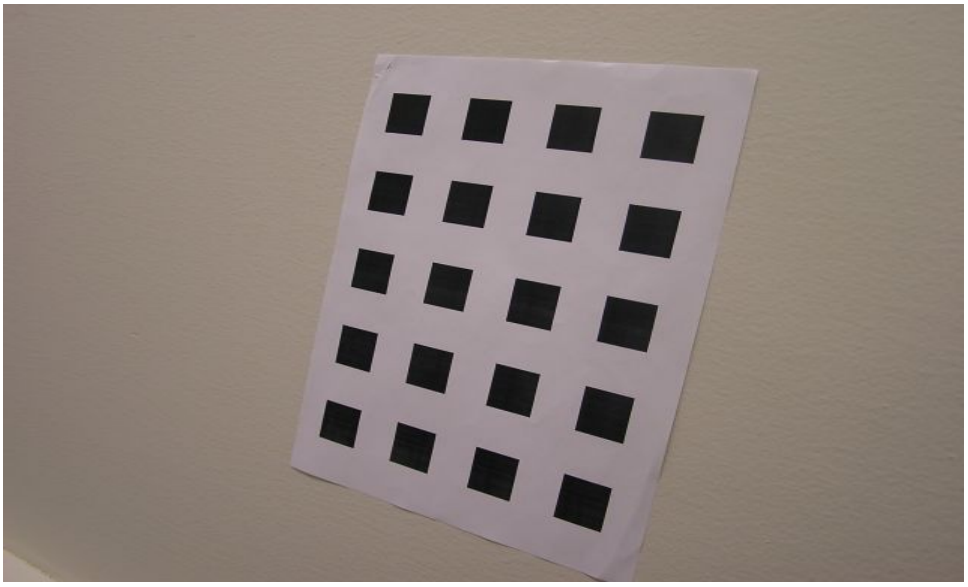

Figure 17: image 1

$$[\mathrm{R}|\mathrm{t}]=\begin{bmatrix} 0.71634 & -0.1686 & 0.48932 & -0.3856 \\ 0.18148 & 0.8993 & 0.3182 & -0.0011 \\ -0.52799 & 0.7999 & 0.6784 & 0.0048 \end{bmatrix}$$

$$[R|t]_{refined}=\begin{bmatrix} 0.7454 & -0.1586 & 0.4882 & -0.3746 \\ 0.1765 & 0.8864 & 0.3054 & -0.0011 \\ -0.5189 & 0.7891 & 0.6684 & 0.0048 \end{bmatrix}$$

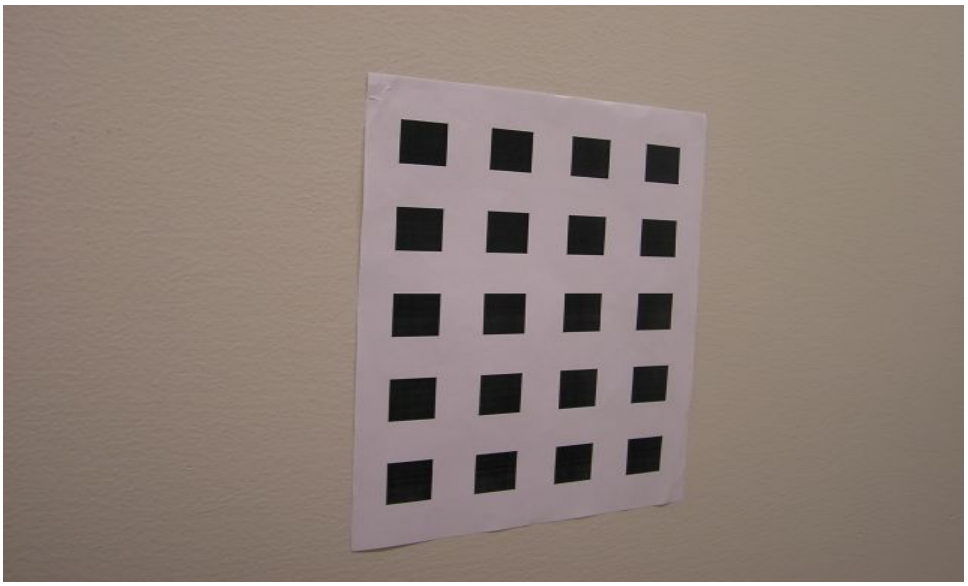

Figure 18: image 2

$$[R|t]=\begin{bmatrix} 0.8124 & -0.39553 & -0.52911 & -0.33856 \\ 0.95567 & 0.97307 & -0.79821 & -0.9089 \\ -0.54383 & 0.71769 & 0.79604 & 0.0046 \end{bmatrix}$$

$$[R|t]_{refined}=\begin{bmatrix} 0.824 & -0.3835 & -0.5111 & -0.3263 \\ 0.9437 & 0.9607 & -0.7821 & -0.91 \\ -0.531 & 0.7064 & 0.7804 & 0.0046 \end{bmatrix}$$

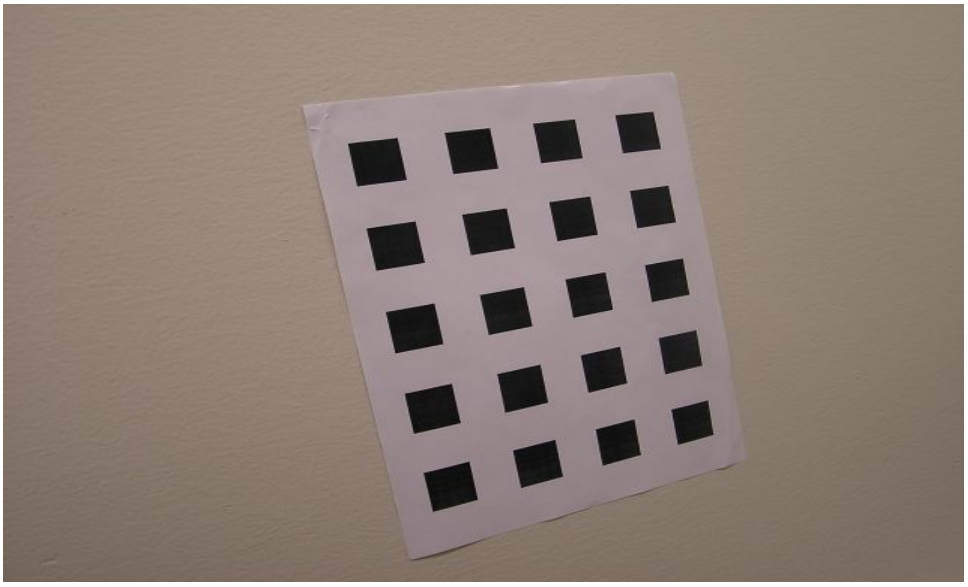

Figure 19: image 3

$$[R|t]=\begin{bmatrix} 0.92957 & 0.1768 & -0.50400 & -0.62233 \\ -0.20574 & 0.10311 & -0.35481 & -0.84995 \\ 0.46358 & 0.12643 & 0.99487 & 0.0051 \end{bmatrix}$$

$$[R|t]_{refined}=\begin{bmatrix} 0.9157 & 0.1768 & -0.50200 & -0.6133 \\ -0.241 & 0.1012 & -0.3413 & -0.83495 \\ 0.4581 & 0.1153 & 0.99487 & 0.0051 \end{bmatrix}$$

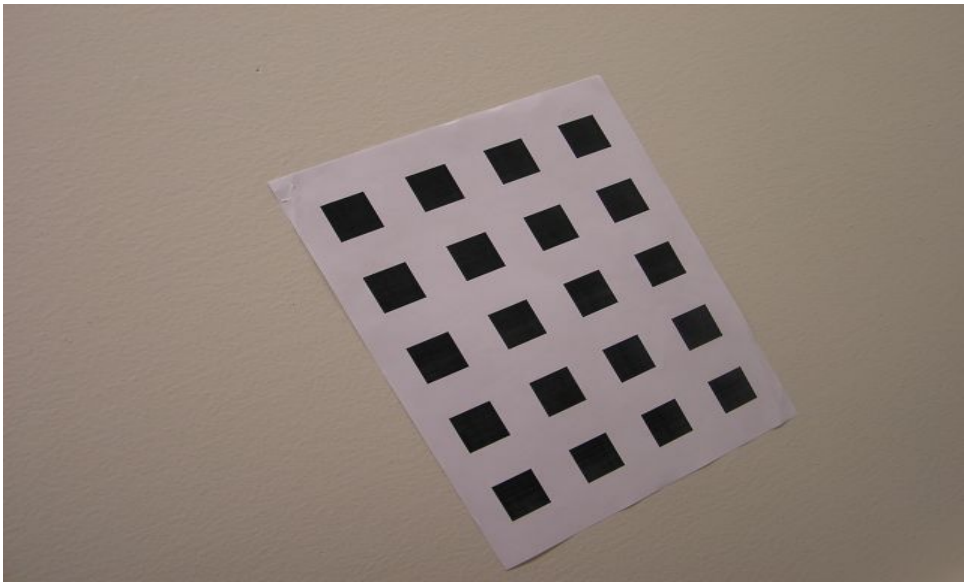

Figure 20: image 4

$$[R|t]=\begin{bmatrix} 0.76300 & 0.39849 & -0.31859 & -0.71822 \\ -0.37840 & 0.82873 & 0.97711 & -0.45864 \\ 0.35762 & 0.58714 & 0.78311 & 0.0048 \end{bmatrix}$$

$$[R|t]_{refined}=\begin{bmatrix} 0.7500 & 0.3849 & -0.3059 & -0.7072 \\ -0.37640 & 0.8172 & 0.96711 & -0.4464 \\ 0.3462 & 0.5614 & 0.78311 & 0.0048 \end{bmatrix}$$ The mean and variance in pixels before and Optimization for the 4 images

Table 1: Mean and variance before and after optimization

Before Optimization mean error	After Optimization Mean error	Before optimization variance	After Optimi
1.83	1.43	0.63	0.42
1.78	1.5	0.68	0.54
1.68	1.32	0.53	0.49
1.63	1.28	0.58	0.43

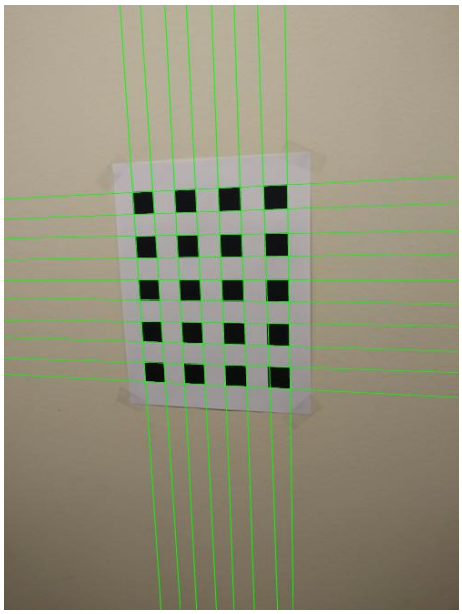

Figure 21: Hough lines

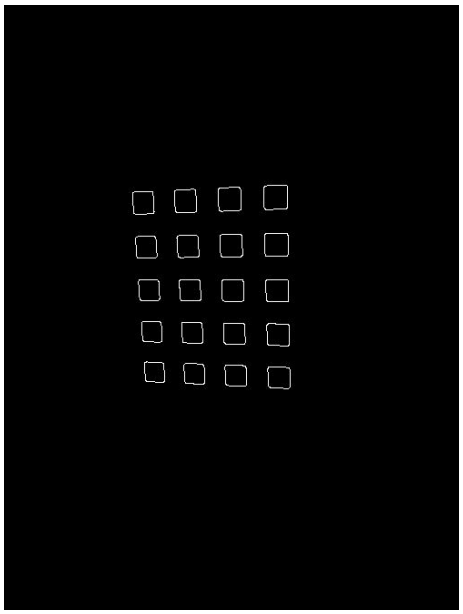

Figure 22: Edges.

Figure 23: corners.

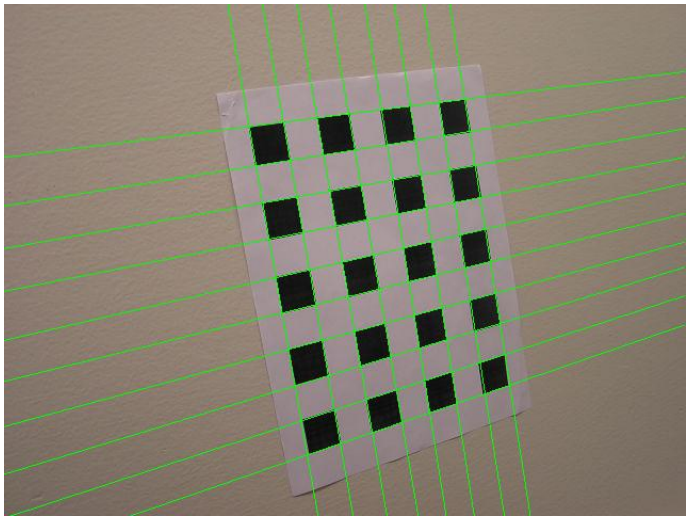

Figure 24: Hough lines

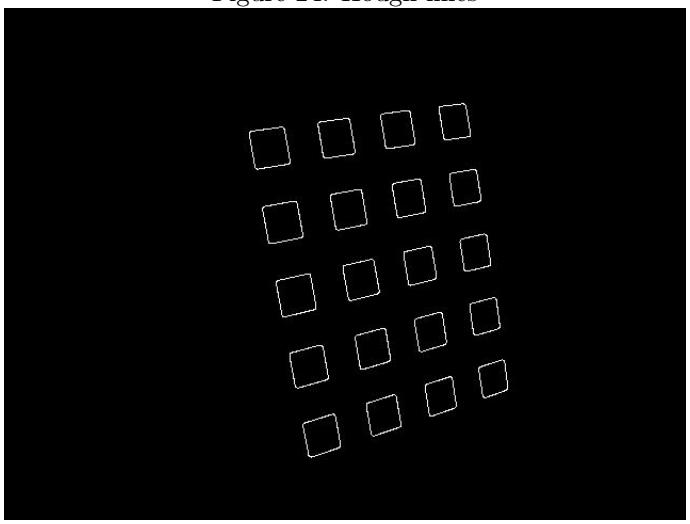

Figure 25: Edges.

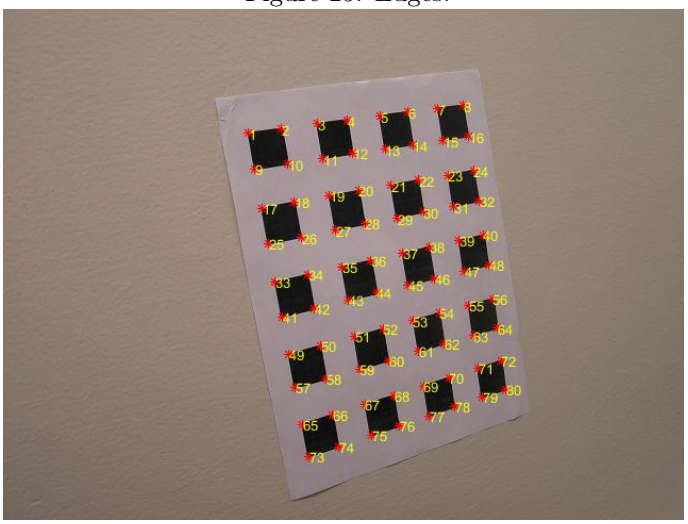

Figure 26: corners.

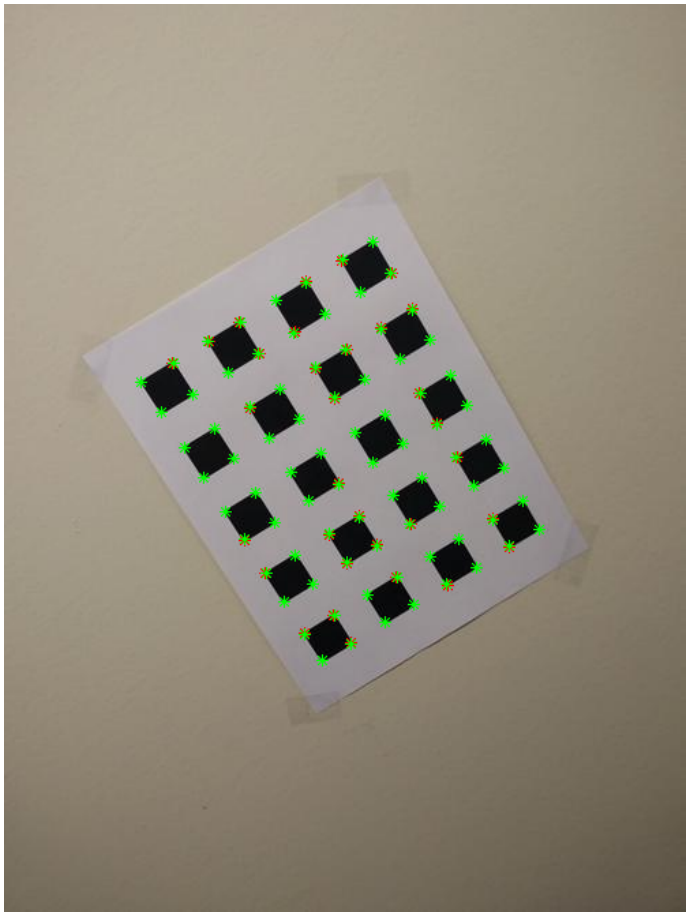

Figure 27: Reprojection with labels

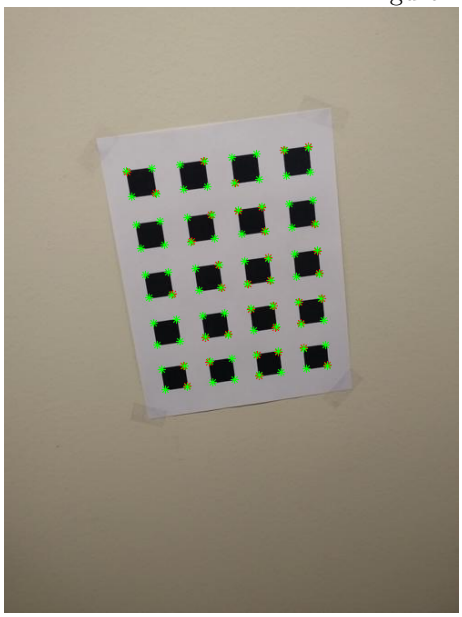

Figure 28: Reprojection with labels.

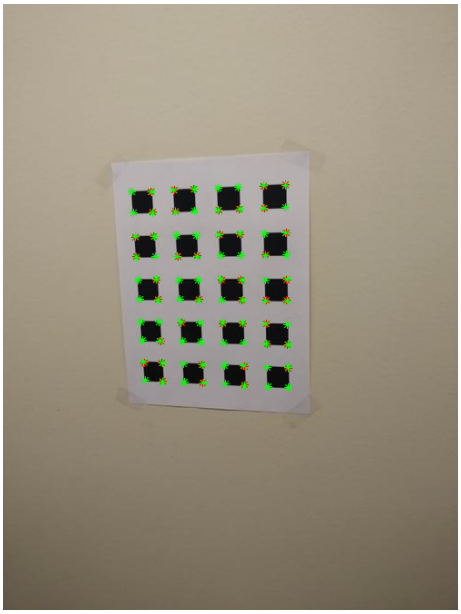

Figure 29: Reprojection with labels

Figure 30: before Reprojection

Figure 31: before Reprojection .

Figure 32: before Reprojection

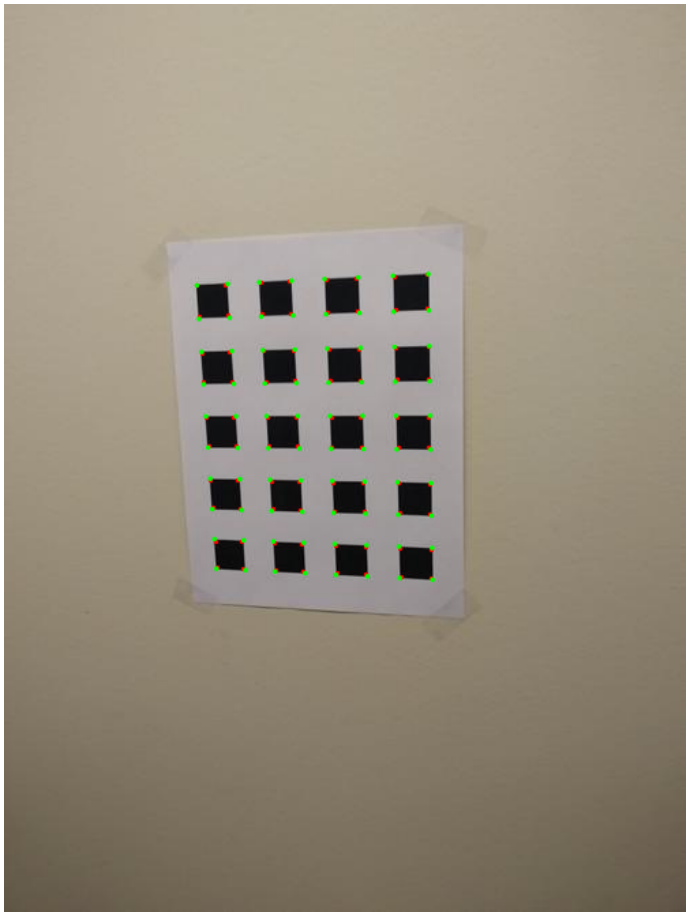

Figure 33: before LM optimization

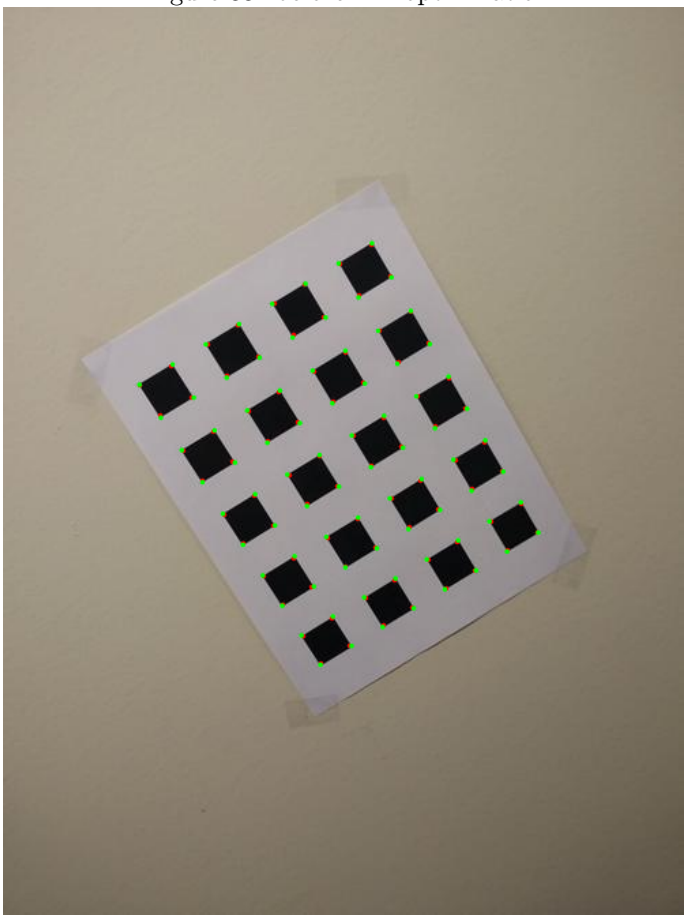

Figure 34: before LM optimization

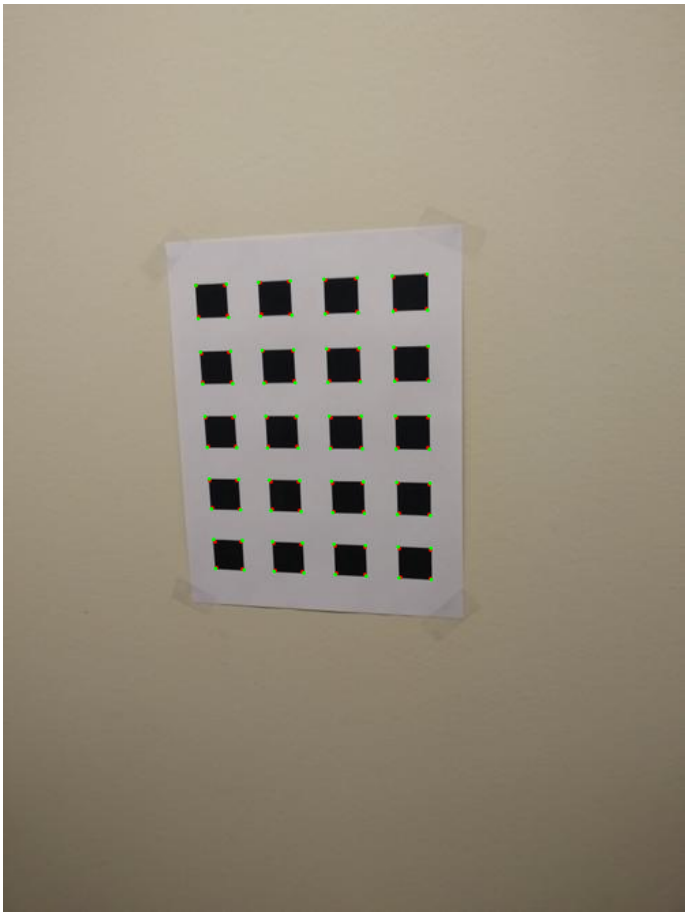

Figure 35: After LM optimization

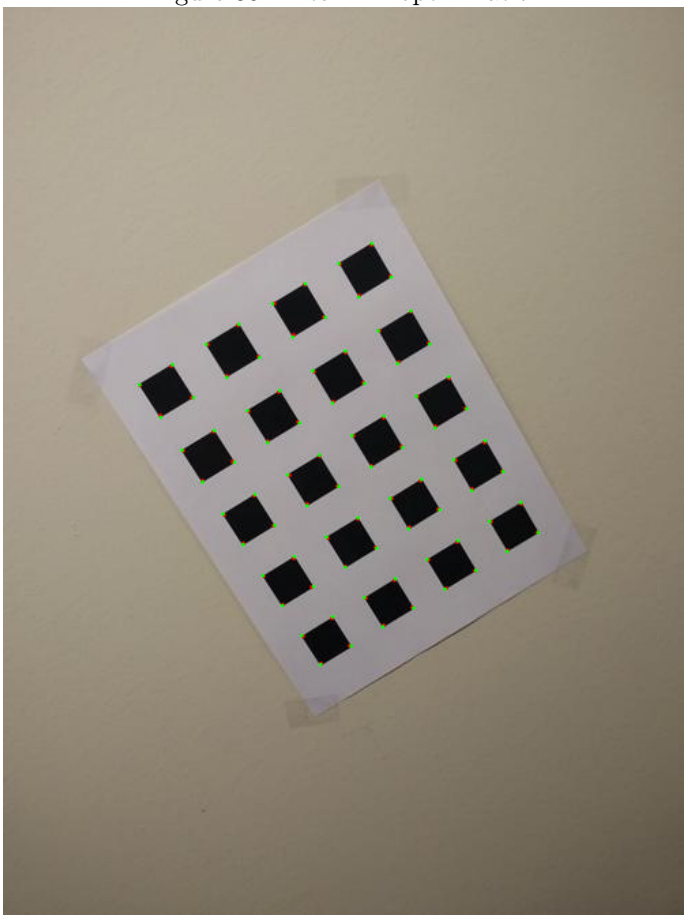

Figure 36: After LM optimization

The internal parameters for the 40 image dataset is $\begin{bmatrix} 547.132 & 0.4667 & 243.74 \\ 0. & 546.68 & 326.905 \\ 0. & 0. & 1. \end{bmatrix}$

The external parameters for Images are

Figure 37: image 1

$$[\mathrm{R}|\mathrm{t}]=\begin{bmatrix} 0.1106 & 0.4129 & 0.31598 & -0.0011 \\ 0.1007 & 0.11209 & -0.24357 & -0.0014 \\ -0.2800 & 0.2097 & 0.1235 & 0.0062 \end{bmatrix}$$

$$[R|t]_{refined}=\begin{bmatrix} 0.1058 & -0.4032 & 0.3043 & -0.0932 \\ 0.1023 & 0.1032 & 0.2054 & -0.0011 \\ -0.219 & 0.2081 & 0.1084 & 0.0058 \end{bmatrix}$$

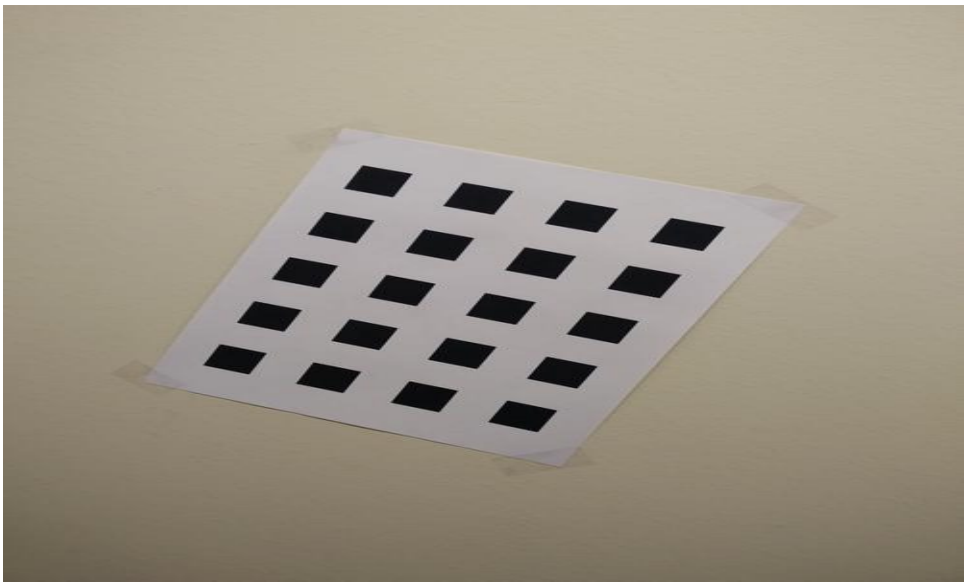

Figure 38: image 2

$$[R|t]=\begin{bmatrix} 0.1159 & -0.45122 & 0.25321 & -0.0058 \\ 0.47349 & 0.1167 & -0.1167 & -0.0015 \\ -0.1535 & 0.1563 & 0.15671 & 0.0059 \end{bmatrix}$$

$$[R|t]_{refined}=\begin{bmatrix} 0.1824 & -0.3835 & -0.2531 & -0.0043 \\ 0.437 & 0.9607 & -0.1101 & -0.001 \\ -0.1531 & 0.7064 & 0.1204 & 0.0049 \end{bmatrix}$$

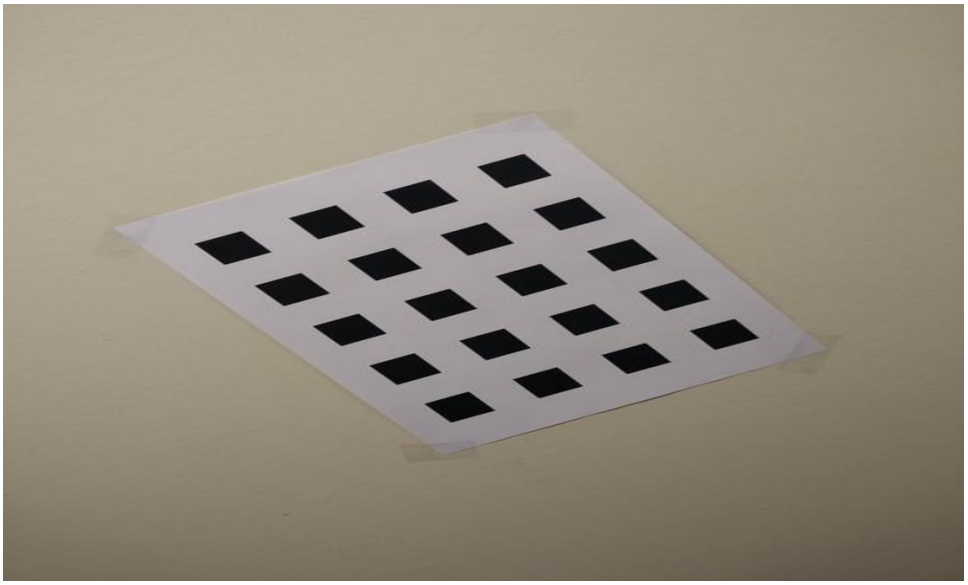

Figure 39: image 3

$$[R|t]=\begin{bmatrix} -0.97579 & -0.58713 & -0.9543 & 0.0014 \\ 0.59225 & -0.96103 & -0.16541 & 0.0048 \\ 0.339 & -0.16654 & 0.12857 & 0.0057 \end{bmatrix}$$

$$[R|t]_{refined}=\begin{bmatrix} -0.9157 & -0.5832 & -0.9021 & 0.0013 \\ 0.541 & -0.9712 & -0.1413 & 0.495 \\ 0.3651 & 0-0.1653 & 0.1108 & 0.0051 \end{bmatrix}$$

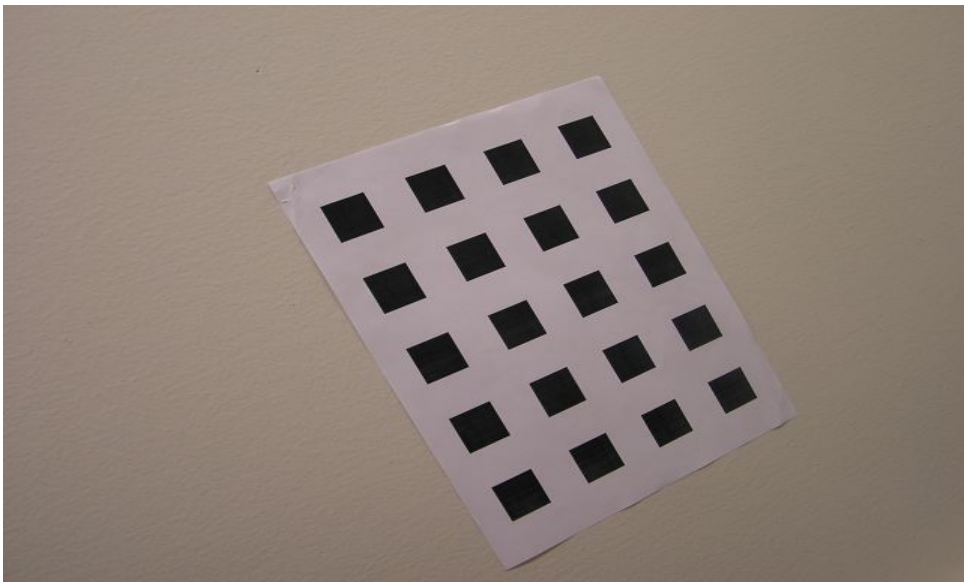

Figure 40: image 4

$$[R|t]=\begin{bmatrix} 0.15078 & 0.25632 & -0.13937 & -0.0015 \\ -0.27615 & 0.14678 & -0.5301 & -0.0018 \\ 0.28187 & 0.35638 & 0.22841 & 0.0069 \end{bmatrix}$$

$$[R|t]_{refined}=\begin{bmatrix} 0.1500 & 0.2249 & -0.1309 & -0.0011 \\ -0.27640 & 0.1172 & 0.56711 & -0.0019 \\ 0.2621 & 0.3514 & 0.2211 & 0.0057 \end{bmatrix}$$ The mean and variance in pixels before and Optimization for the 4 images

Table 2: Mean and variance before and after optimization

Before Optimization mean error	After Optim Mean error	Before optimization variance	After Optim varianc
1.32	0.95	0.43	0.32
1.68	1.15	0.58	0.43
1.55	1.23	0.61	0.45
1.43	1.18	0.56	0.42

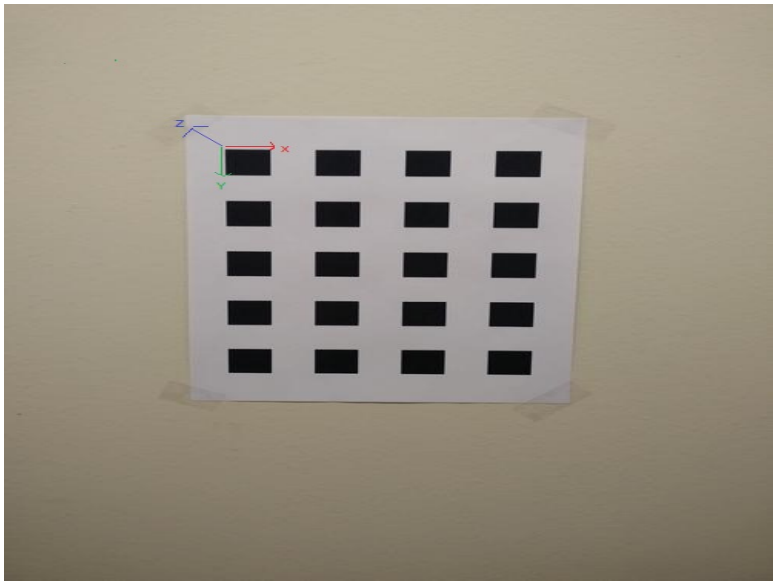

Figure 41: The world frame is shown with coordinates with Z out of the image and X to left and Y towards bottom, all corners are assumed to be seperated by 1 inch.

Radial distortion parameters are K1=-0.15438,k2=0.5239
Max reprojection error=1.43pixels
for my own dataset k1=-0.2345,k2=0.4532
max reprojection error=1.52 pixels.

5 Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 20 12:35:16 2016

@author: dwarak
"""
import cv2
```

```
import numpy as np
import glob
import matplotlib.pyplot as plt
from scipy.optimize import leastsq
import os
import time
import pickle
####imports completed here

"""
This file contains functions to implement
Zhangs camera calibration procedure and
calculate eternal and internal parameters
also contains many helper functions and
canny edge detector and hough line fitting and
radial distortion parameters
"""

font = cv2.FONT_HERSHEY_SIMPLEX
def fit_hough_lines(image):
        ##returns hough lines
    img = cv2.imread(image)
    gray        =        cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    edges        =         cv2.Canny(gray,255*1.5,255)
    cv2.imwrite('houghedges'+image+'.jpg',edges)
    lines=cv2.HoughLines(edges,1,np.pi/180,50)

    return lines

def print_lines(lines,image):
        ##draws hough lines on the image
    new_image=cv2.CloneMat(image)
    for x in range(0,len(lines)):
        for r,t in lines[x]:
            a=np.cos(t)
            b=np.sin(t)
            x0=a*r
            y0=b*r
            x1=int(x0+1000*(-b))
            y1=int(y0+1000*(a))
            x2=int(x0-1000*(-b))
            y2=int(y0-1000*(a))
            cv2.line(new_image,(x1,y1),(x2,y2),(0,255,0),1)
    return new_image
def checker_lines(lines,checker_pattern,image):
    thetas=np.zeros((len(lines)))
    for i,x in enumerate(lines):
        print lines
        for r,t in lines[i]:
            thetas[i]=t
```

```
thetas-=np.pi/2
linebadness=range(len(thetas))
hldxs=map(int,np.where(abs(thetas) <np.pi/4)[0])
horz_lines=[lines[i] for i in hldxs]
hl_badness=[linebadness[i] for i in hldxs]
vldxs=map(int,np.where(abs(thetas)>=np.pi/4)[0])
vert_lines=[lines[i] for i in vldxs]
vline_badness=[linebadness[i] for i in vldxs]
assert (len(hldxs) +len(vldxs)==len(thetas))

hlineandbadness=zip(hl_badness,horz_lines)
#horz_lines=get_mul(horz_lines)
horz_lines=sorted(horz_lines,key=lambda temp_var:temp_var[1][0]*np.sin(temp_var
hlineandbadness=sorted(hlineandbadness,key=lambda temp_var:temp_var[1][0]*np.si
vlineandbadness=zip(vline_badness,vert_lines)
vert_lines=sorted(vert_lines,key=lambda temp_var:temp_var[0]*np.cos(temp_var[1]
vlineandbadness=sorted(vlineandbadness,key=lambda temp_var:temp_var[1][0]*np.co
def group_lines(hl_with_badness,sinCos,minsep):
    uniquelines=[]
    uniquelinesbadness=[]
    hvline=hl_with_badness[0][1]
    uniquelines.append(hvline)
    badness=hl_with_badness[0][0]
    uniquelinesbadness.append(badness)
    for i in xrange(1,len(hl_with_badness)):
        hvlines=hl_with_badness[i][1]
        badness=hl_with_badness[i][0]
        linehc=get_hough_line_hc(hvline)

        curlinehc=get_hough_line_hc(hvline)
        uniquelinehc=get_hough_line_hc(uniquelines[-1])
        corner=np.cross(curlinehc,uniquelinehc)
        corner=convert_to_homogeneous(corner)
        if (((corner[0]>=0) and (corner[0]<checker_pattern[1])) and ((corner[1]
            if uniquelinesbadness[-1]>badness:
                uniquelines[-1]=hvlines
                uniquelinesbadness[-1]=badness;
            continue
        uniquelinedist=uniquelines[-1][0]*sinCos(uniquelines[-1][1])
        curdist=hvlines[0]*sinCos(hvlines[1])
        deltadist=abs(uniquelinedist-curdist)
        if deltadist<=minsep:
            if uniquelinesbadness[-1]>badness:
                uniquelines[-1]=hvlines
                uniquelinesbadness[-1]=badness;
            continue
        uniquelines.append(hvlines)
        uniquelinesbadness.append(badness)
        sepdist=deltadist
    return (uniquelines,sepdist)
```

```
    minsep=1.0
    is_done=False
    while is_done==False:
        (uniquehlines,hsepdist)=group_lines(hlineandbadness,np.sin,minsep)
        (uniquevlines,vsepdist)=group_lines(vlineandbadness,np.cos.minsep)
        if ((len(uniquehlines)==2*checker_pattern[0]) and (len(uniquevlines)==2*che
            is_done=True
        elif ((len(uniquehlines)<2*checker_pattern[0]) or (len(uniquevlines)<2*chec
            is_done=True
            assert(0)
        else:
            minsep+=1
    assert (len(uniquehlines)>=2*checker_pattern[0])
    assert (len(uniquevlines)>=2*checker_pattern[1])
    bestlines=uniquehlines+uniquevlines
    sepdist=int((hsepdist+vsepdist)/2.0)
    return (bestlines,sepdist)


def get_mul(lines):
    x=[]
    for i in range(len(lines)):
        x.append(lines[0]*np.sin(lines[1]))
   # x=np.array(x)
    return x


def Reshape_corners(cornerset,dim):
    n_rows=dim[0]
    n_cols=dim[1]
    corners=[[cornerset[i*n_cols+j] for j in range(n_cols)] for i in range(n_rows)]
    return corners
def get_hough_line_hc(line):
    ro=line[0]
    theta=line[1]
    pt1=np.array([ro*np.cos(theta),ro*np.sin(theta),1.0])
    pt2=np.array([pt1[0]+100*np.sin(theta),pt1[1]-100*np.cos(theta),1.0])
    line_hc=np.cross(pt1,pt2)
    return line_hc

def print_corners(corners,image):
    for i in range(len(corners)):
        point=tuple(map(int,corners[i]))
        cv2.circle(image,point,3,(0,0,255),-1)
    return image
def print_labels(corners,image):
    rows=len(corners)
    cols=len(corners[0])
    corner_count=0
```

```
    for row in range(rows):
        for col in range(cols):
            point=tuple(map(int,corners[row][col]))
            cv2.circle(image,point,3,(0,0,255),-1)
            cv2.PutText(image,str(corner_count),point,font,(255,0,0))
            corner_count+=1
    return image


def refine_corners(image,corners,window):
        ##This function hels to refine corners to subpixel accuracy by using
        ##opencv cornersubpix function
    corner_1d=[corner for i in corners for corner in i]
    criteria=(cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 0.01)
    corner_1d=cv2.cornerSubPix(image,corner_1d,(window/2,window/2),(2,2),criteria)
    corner_2d=Reshape_corners(corner_1d,(len(corners),len(corners[0])))
    return corner_2d

def convert_to_homogeneous(point):
    point[0]/=point[2]
    point[1]/=point[2]
    point[2]/=point[2]
    return point

def find_corners(lines,dist,dim):
        ##This functin returns corners and takes input as lines and dist sepeartion

    thetas=np.array([lines[i][1] for i in range(len(lines))])
    thetas=-np.pi/2
    horz_is=np.where(abs(thetas)<np.pi/4)[0]
    horz_lines=[lines[i] for i in horz_is]
    vldxs=np.where(abs(thetas)>=np.pi/4)[0]
    vert_lines=[lines[i] for i in vldxs]
    assert(len(horz_is)+len(vldxs)==len(thetas))

    n_hlines=len(horz_lines)
    n_vlines=len(vert_lines)
    assert(n_hlines==2*dim[0])
    assert(n_vlines==2*dim[1])
    ##intersection of lines here
    corners=[[() for j in range(n_vlines)] for i in range(n_hlines)]
    horz_lines=sorted(horz_lines,key=lambda temp_var:temp_var[0] *np.sin(temp_var[1
    vert_lines=sorted(vert_lines,key=lambda temp_var:temp_var[0]*np.cos(temp_var[1]
    for hldx in xrange(n_hlines):
        h_line_hc=get_hough_line_hc(horz_lines[hldx])
        for vldx in xrange(n_vlines):
            v_line_hc=get_hough_line_hc(vert_lines[vldx])

            corner=np.cross(h_line_hc,v_line_hc)
            corner=convert_to_homogeneous(corner)
```

```
            corners[hldx][vldx]=(corner[0],corner[1])
    return corners

def gradient_descent(x,y,p):
    ##computes gradinet descent
    new_p=p
    m=len(x)
    for i in range(100):
        grad=1.0/m*sum(compute_difference(y,pp(x,new_p)))

        temp=new_p-0.001*grad
        new_p=temp
    print new_p
def pp(x,p):
        ##helper function to multipy x in 4d with P 3*4
    l=[]
    print len(x)
    for i in range(len(x[0])):
        point=convert_to_4d(x[0][i])
        new_x=multiply_x(p,point)
        k=[0,0]
        k[0]=new_x[0]
        k[1]=new_x[1]
        print k

        l.append(k)
    return l

def convert_to_4d(x):
        ##convert x from 2,1 to 4,1

    point=np.zeros((4,1),dtype='float')
    point[0]=x[0]
    point[1]=x[1]
    point[3]=1

    return point

def multiply_x(p,x):
        #rturns dot product of p and x
    return np.dot(p,x)

def compute_difference(y,x):
        ##compute the cost functin
        ## y is x_ij and x is x_hat_ij
    print
    print x
    time.sleep(3)
    diff=[]
    for i in range(len(y)):
```

```
        temp=y[i]-x[i]

        diff.append(temp)
    print diff
    return diff
def load_data(folder):
        ##load data
    c=[]
    for i in os.listdir(folder):
        data=cv2.imread(folder+'/'+i,dtype='float')
        c.append(data)
    return c

def calculate_radial_pixel(p,k,point):
        ##returns radial pixel from normal pixel
        ##radial coordinate estimate
    m=p[:,:3]
    t=p[:,3:4]
    k1=k[:,:1]
    k2=k[:,1:2]
    pp=np.dot(m,t)
    pp=convert_to_homogeneous(pp)
    point=convert_to_homogeneous(point)
    r=((point[0]-pp[0])**2)+((point[1]-pp[1])**2)
    temp_var=np.dot(k1,r)+np.dot(k2,r**2)
    temp_x=np.dot((point[0]-pp[0]),temp_var)
    temp_y=np.dot((point[1]-pp[1]),temp_var)
    x_rad=point[0]+temp_x
    y_rad=point[1]+temp_y


    return np.array([x_rad,y_rad,1.0])


def convert_rodregus_matrix(matrix):
        ##convert the rodregus matrix to normal form
    point=np.zeros((3,),dtype='float')
    point[0]=matrix[2][1]
    point[1]=matrix[0][2]
    point[2]=matrix[1][0]
    return point
def convert_to_rodregus(point):
        ##convert the point to rodregus marix 3*3
    new_matrix=np.zeros((3,3),dtype='float')
    new_matrix[0][1]=-point[2]
    new_matrix[0][2]=point[1]
    new_matrix[1][0]=point[2]
    new_matrix[1][2]=-point[0]
    new_matrix[2][0]=-point[1]
    new_matrix[2][1]=point[0]
```

```
    return new_matrix


def compute_homography(pointa,pointb):
    """
    computing the homography with the two points given
    """
    A=np.zeros((8,8))
    #make A with dimension of 8*8
    b=np.zeros((1,8))
    #make b with dimension of (1*8)
    for i in range(len(pointa)):
        #iterating for the length of pointa and calculating the correspondng points
        A[2*i]=[pointa[i][0],pointa[i][1], 1, 0, 0, 0, -(pointa[i][0]*pointb[i][0])
        A[2*i + 1]=[0, 0, 0, pointa[i][0], pointa[i][1], 1, -(pointa[i][0]*pointa[i
        b[0][2*i]=pointb[i][0]
        b[0][2*i + 1]=pointb[i][1]

    H_temp=np.dot(np.linalg.pinv(A),b.T) #taking the transpose of b here bcoz we ha
    H=np.zeros((3,3)) #H is 3*3 matrix


    H[0][0]=H_temp[0]
    H[0][1]=H_temp[1]
    H[0][2]=H_temp[2]
    H[1][0]=H_temp[3]
    H[1][1]=H_temp[4]
    H[1][2]=H_temp[5]
    H[2][0]=H_temp[6]
    H[2][1]=H_temp[7]
    H[2][2]=1
    #making the corresponding values for H and returing the H matrix
    return H
def gen_world_coordinates(sep=1):
        coord=[]
        #generating world gen_world_coordinates
        x=0
        y=0
        for i in range(11):

                for j in range(9):
                        y+=1
                        coord.append([x,y])
                x+=1

        return coord

def estimate_r(k,h):
        #returns r and t
        k_inv=np.linalg.inv(k)
```

```
        h1=h[1,:]
        h2=h[2,:]
        h3=h[3,:]
        t=np.dot(k_inv,h3)
        r1=np.dot(k_inv,h1)
        r2=np.dot(k_inv,h2)
        r3=np.cross(r1,r2)
        return r1,r2,r3,t

def get_v(h1,h2):
    return np.array([h1[0]*h2[0],h1[0]*h2[1]])
def solve_for_w(H):
    h1=H[:,0]
    h2=H[:1]
    h1_v=get_v(h1,h2)
    h2_v=get_v(h1,h1)
    h11_v=get_v(h2,h2)
    v=[h1_v,h2_v,h11_v]
    u,d,v=np.linalg.svd(v)
    mineigval=v[d.argmin()]
    w[0,0]=mineigval[0]
    w[0,1]=mineigval[1]
    w[0,2]=mineigval[2]
    w[1,0]=w[0,1]
    w[1,1]=mineigval[3]
    w[1,2]=mineigval[4]
    w[2,0]=2[0,2]
    w[2,1]=w[1,2]
    w[2,2]=mineigval[5]
    return w
def solve_for_k(w):
    y=(w[0,0]*w[0,2]-w[0,0]*w[1,2]/w[0,0]*w[1,1]-w[0,1]*w[0,1])
    l=w[2,2]-(w[0,2]*w[0,2] +y*(w[0,1]*w[0,2]-w[0,0]*w[1,2]))/w[0,0]
    alpha_x=np.sqrt(l/w[0,0])
    alpha_y=np.sqrt(l*w[0,0]/(w[0,0]*w[1,1]-w[0,1]*w[0,1]))
    s=-(w[0,1]*alpha_x*alpha_x*alpha_y)/l
    x0=(s*y)/alpha_y-(w[0,2]*alpha_x*alpha_x)/l
    k=np.zeros((3,3),dtype=float)
    k[0,0]=alpha_x
    k[0,1]=s
    k[0,2]=x0
    k[1,0]=0
    k[1,1]=alpha_y
    k[1,2]=y
    k[2,2]=1

    return k
def reproject_corners(r1,r2,r3,t,coords):
    data=[]
    p=np.zeros((3,4),dtyep='float')
```

```
    p=np.transpose(p)
    p[1,:]=r1
    p[2,:]=r2
    p[3,:]=r3
    p[4,:]=t
    p_inv=np.linalg.pinv(p)
    for i in range(len(coords)):
        coord=convert_to_4d(coords[i])
        data.append(np.dot(p_inv,coord))
    return data,p

sep=1
H=[]
dataset=[]
image_files=load_data('Dataset1')
check_dim=(5,4)
for image in image_files:
        image_dim=(image.shape[0],image.shape[1])
        lines=fit_hough_lines(image)
        new_img=print_lines(lines,image)
        cv2.imwrite('new_image_with_corners.jpg',new_img)
        corners=find_corners(lines,image_dim,check_dim)
        corners=refine_corners(image,corners,check_dim)
        new_image=print_labels(corners,image)
        cv2.imwrite('new_image_with_corners_labels.jpg',new_image)
        h=compute_homography(corners,gen_world_coordinates)
        H.append(h)
        w=solve_for_w(H)
        k=solve_for_k(w)
        r1,r2,r3,t=estimate_r(k,h)
        corners_rep,p=reproject_corners(r1,r2,r3,t,k,gen_world_coordinates)
        new_image=print_corners(corners,image)
        new_p=gradient_descent(corners_rep,corners,p)
```