```
from scipy import optimize

import cv2

import numpy as np


#Function used to create a line from two points

def create_line(x1,y1,x2,y2):

    p1 = np.array([x1,y1,1])

    p2 = np.array([x2,y2,1])

    line = np.cross(p1,p2)

    return line

#Function to find points on hough line

def create_point(rh,th):

    a = np.cos(th)

    b = np.sin(th)

    x0 = a*rh

    y0 = b*rh

    x1 = int(x0 + 1000*(-b))

    y1 = int(y0 + 1000*(a))

    x2 = int(x0 - 1000*(-b))

    y2 = int(y0 - 1000*(a))

    return x1,y1,x2,y2

#Function to extract hough lines from an image

def houghlines(image, N):

    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

    edge = cv2.Canny(gray, 400, 300)

    cv2.imwrite('C:/Users/namra/Desktop/Fall 2018/ECE 661/hw8/canny/Canny_%i.jpg'%(N), edge)

    lines = cv2.HoughLines(edge,1,np.pi/180,50)

    lines = lines[0]

    new_image = image.copy()
```

```
theta = np.array([lines[i][1] for i in range(len(lines))])

theta = theta - np.pi/2

hor_loc=np.where(abs(theta) < np.pi/4)

hor_lines = []

for i in hor_loc[0]:

   hor_lines.append(lines[i])


ver_loc=np.where(abs(theta) > np.pi/4)

ver_lines = []

for j in ver_loc[0]:

   ver_lines.append(lines[j])


hp = []

hp.append(hor_lines[0])

nna = 1

c = 0

for a in range (1, len(hor_lines)):

   for b in range (0,nna):

      if (hor_lines[a][0] < hp[b][0] + 9) and (hor_lines[a][0] > hp[b][0] - 9):

         break

      else:

         c = c  + 1

   if (c==nna)and(c!= 0):

      hp.append(hor_lines[a])

      nna = nna + 1

   c = 0

   if (nna == 10):

      break

hp = sorted(hp ,key=lambda hp:hp[0]*np.sin(hp[1]))
```

```
for rh, th in hp:
   x1,y1,x2,y2 = create_point(rh,th)
   cv2.line(new_image,(x1,y1),(x2,y2),(0,255,0),2)



vp = []
vp.append(ver_lines[0])
nna = 1
c = 0
for a in range (1, len(ver_lines)):
   for b in range (0,nna):
      if (ver_lines[a][0] < vp[b][0] + 9) and (ver_lines[a][0] > vp[b][0] - 9):
         break
      else:
         c = c  + 1
   if (c==nna)and(c!= 0):
      vp.append(ver_lines[a])
      nna = nna + 1
   c = 0
   if (nna == 8):
      break

vp = sorted(vp ,key=lambda vp:vp[0]*np.cos(vp[1]))

for rh, th in vp:
   x1,y1,x2,y2 = create_point(rh,th)
   cv2.line(new_image,(x1,y1),(x2,y2),(0,255,0),2)
```

```
    cv2.imwrite('C:/Users/namra/Desktop/Fall 2018/ECE
661/hw8/h_lines/vertical_%i.jpg'%(N),new_image)


    return hp,vp
#Function to plot corners in the fixed image
def createfixedcorners(image, h_line, v_line,N):
    img = image.copy()
    corners = []
    font = cv2.FONT_HERSHEY_DUPLEX
    i = 1
    for h in h_line:
        x1,y1,x2,y2 = create_point(h[0],h[1])
        line1 = create_line(x1,y1,x2,y2)
        for v in v_line:
            x1,y1,x2,y2 = create_point(v[0],v[1])
            line2 = create_line(x1,y1,x2,y2)
            c = np.cross(line1, line2)
            c = c/c[2]
            corners.append(c)
            cv2.putText(img,'%i'%(i),(c[0],c[1]), font, 0.5,(255,0,0),1)
            i = i + 1
    cv2.imwrite('C:/Users/namra/Desktop/Fall 2018/ECE 661/hw8/corners/corners_%i.jpg'%(N), img)
    return corners, img
#Function to plot corners in am image
def createcorners(image, h_line, v_line,N):
    img = image.copy()
    corners = []
    font = cv2.FONT_HERSHEY_DUPLEX
    i = 1
```

```
    for h in h_line:
        x1,y1,x2,y2 = create_point(h[0],h[1])
        line1 = create_line(x1,y1,x2,y2)
        for v in v_line:
            x1,y1,x2,y2 = create_point(v[0],v[1])
            line2 = create_line(x1,y1,x2,y2)
            c = np.cross(line1, line2)
            c = c/c[2]
            corners.append(c)
            cv2.circle(img,(c[0],c[1]), 2, (0,0,255), thickness = 3 )
            cv2.putText(img,'%i'%(i),(c[0],c[1]), font, 0.5,(255,0,0),1)
            i = i + 1
    cv2.imwrite('C:/Users/namra/Desktop/Fall 2018/ECE 661/hw8/corners/corners_%i.jpg'%(N), img)
    return corners, img
#Function to calculate homograohy
def homography (pt_i, pt_f):
    T = np.zeros((2*len(pt_i),8))
    s = np.zeros((2*len(pt_i),1))
    for i in range (len(pt_i)):
        T[2*i]= [pt_i[i][0],pt_i[i][1],1, 0, 0, 0, (-1*pt_i[i][0]*pt_f[i][0]), (-1*pt_i[i][1]*pt_f[i][0])]
        T[2*i+1] = [0, 0, 0, pt_i[i][0], pt_i[i][1], 1, (-1*pt_i[i][0]*pt_f[i][1]), (-1*pt_i[i][1]*pt_f[i][1])]
        s [2*i] = pt_f[i][0]
        s [2*i+1] = pt_f[i][1]

    H = np.zeros((3,3))
    inv_T =np.matmul(np.linalg.inv(np.matmul(np.transpose(T),T)),np.transpose(T))
    h = np.matmul(inv_T, s)
    H[0] = h[0:3,0]
    H[1][0] = h[3][0]
```

```
    H[1][1] = h[4][0]

    H[1][2] = h[5][0]

    H[2][0] = h[6][0]

    H[2][1] = h[7][0]

    H[2][2] = 1

    return H

#Function to calculate world coordinates

def worldcord():

    corner = []

    for i in range (0,10):

        for j in range (0,8):

            corner.append([j, i, 1])


    return corner

#Function to calculate vector V

def create_v(H,k,i,j):

    H = np.transpose(H[k])

    v = np.array([H[i][0]*H[j][0], H[i][0]*H[j][1] + H[i][1]*H[j][0], H[i][1]*H[j][1], H[i][2]*H[j][0] +
H[i][0]*H[j][2], H[i][2]*H[j][1] + H[i][1]*H[j][2], H[i][2]*H[j][2]])

    return v

#Function to calculate absolute conic

def calc_w(H):

    k = 0

    i = 0

    j = 1

    v12 = create_v(H,k,i,j)

    i = 0

    j = 0

    v11 = create_v(H,k,i,j)
```

```
i = 1
j = 1
v22 = create_v(H,k,i,j)
V =  np.array([v12, (v11-v22)])
print(V.shape)

for k in range (1, len(H)):
   i = 0
   j = 1
   v12 = np.array([create_v(H,k,i,j)])
   i = 0
   j = 0
   v11 = np.array([create_v(H,k,i,j)])
   i = 1
   j = 1
   v22 = np.array([create_v(H,k,i,j)])
   V = np.append(V , v12, axis = 0 )
   V = np.append(V , v11-v22, axis = 0 )
print(V.shape)
w = np.zeros((3,3))
u,d,vt = np.linalg.svd(V)

v = np.transpose(vt)

print(v[:, -1])

w[0][0] = v[0][5]
w[0][1] = v[1][5]
w[1][1] = v[2][5]
```

```
    w[0][2] = v[3][5]
    w[1][2] = v[4][5]
    w[2][2] = v[5][5]
    w[1][0] = w[0][1]
    w[2][0] = w[0][2]
    w[2][1] = w[1][2]


    return w
#Function to calculate intrinsic parameters
def calc_k (w):
    x_0 = (w[0][1]*w[0][2] - w[0][0]*w[1][2]) / (w[0][0]*w[1][1] - np.square(w[0][1]))
    lamb = w[2][2] - ((np.square(w[0][2]) + x_0*(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]-np.square(w[0][1])))
    s = -1* (w[0][1]*np.square(alpha_x)*alpha_y)/lamb
    y_0 = ((s*x_0)/alpha_y) - ((w[0][2]*np.square(alpha_x))/lamb)

    K = np.zeros((3,3))
    K[0][0] = alpha_x
    K[0][1] = s
    K[0][2] = x_0
    K[1][1] = alpha_y
    K[1][2] = y_0
    K[2][2] = 1
#    print (np.dot(np.transpose(np.linalg.inv(K)),np.linalg.inv(K)))

    return K
#Fucntion to calculate extrinsic parameters
```

```
def calcRT(K, H):
    r1 = np.matmul(np.linalg.inv(K), H[:,0])
    r2 = np.matmul(np.linalg.inv(K), H[:,1])
    t = np.matmul(np.linalg.inv(K), H[:,2])
    P = np.zeros((3,3))

    P[:,0] = r1/np.linalg.norm(r1)
    P[:,1] = r2/np.linalg.norm(r1)
    P[:,2] = t/np.linalg.norm(r1)

    Z = np.zeros((3,4))

    Z[:,0] = r1/np.linalg.norm(r1)
    Z[:,1] = r2/np.linalg.norm(r1)
    Z[:,2] = np.cross (r1/np.linalg.norm(r1), r2/np.linalg.norm(r1))
    Z[:,3] = t/np.linalg.norm(r1)

    print (Z)

    return P
#Function to reproject corners
def reproject(K, P, img, wc, N, corner):
    new_cor = []
    for i in range (0, len(wc)):
        wc_new = np.matmul(np.linalg.inv(np.matmul(K,P)),wc[i])
        wc_new = wc_new/wc_new[2]
        new_cor.append(wc_new)
        cv2.circle(img,(int(wc_new[0]),int(wc_new[1])), 2, (0,0,255), thickness = 2 )
        cv2.circle(img,(int(corner[i][0]),int(corner[i][1])), 2, (0,255,0), thickness = 2 )
```

```
    cv2.imwrite('C:/Users/namra/Desktop/Fall 2018/ECE 661/hw8/reproject/reproject_%i.jpg'%(N),img)
    E_dist = []
    for b in range(len(wc)):
        t = np.sqrt(np.square(corner[b][0]-new_cor[b][0]) + np.square(corner[b][1]-new_cor[b][1]))
        E_dist.append(t)
    mean = sum(E_dist)/len(E_dist)
    print ('Mean_%i : %f'%(N,mean))
    var = sum(np.square(E_dist - mean))/len(E_dist)
    print('Variance_%i : %f'%(N,var))
#Fucntion to calculate extrinsic parameters while using LM
def rod_P(K, H):
    r1 = np.matmul(np.linalg.inv(K), H[:,0])
    r2 = np.matmul(np.linalg.inv(K), H[:,1])
    t = np.matmul(np.linalg.inv(K), H[:,2])

    R1 = r1/np.linalg.norm(r1)
    R2 = r2/np.linalg.norm(r1)
    R3 = np.cross(R1,R2)
    T1 = t/np.linalg.norm(r1)

    R = np.zeros((3,3))
    R[:,0] = R1
    R[:,1] = R2
    R[:,2] = R3

    P = np.zeros((3,3))
    P[:,0] = R[:,0]
    P[:,1] = R[:,1]
```

```
    P[:,2] = T1
    return P
#Function to calculate cost fucntion and jacobian matrix
def func(h):
    error = 0
    j = 0
    f_x = []
    f_y = []
    xrw = []
    yrw = []
    error = []
    Jac = np.zeros((2*len(wc),len(h)))
    for j in range(0, len(wc)):
        x = wc[j][0]
        y = wc[j][1]
        xrw.append(cor[j][0])
        yrw.append(cor[j][1])

        num_x = h[0]*x + h[1]*y + h[2]
        den = h[6]*x + h[7]*y + h[8]
        x_new = np.divide(num_x,den)

        num_y = h[3]*x + h[4]*y + h[5]
        y_new = np.divide(num_y,den)

        f_x.append(x_new)
        f_y.append(y_new)

        Jac [j][0]= -x/den
```

```
        Jac [j][1]= -y/den
        Jac [j][2]= -1/den
        Jac [j][3]= 0
        Jac [j][4]= 0
        Jac [j][5]= 0
        Jac [j][6]= x*(num_x)*np.power(den,-2)
        Jac [j][7]= y*(num_x)*np.power(den,-2)
        Jac [j][8]= num_x*np.power(den,-2)

        Jac [j+1][0]= 0
        Jac [j+1][1]= 0
        Jac [j+1][2]= 0
        Jac [j+1][3]= -x/den
        Jac [j+1][4]= -y/den
        Jac [j+1][5]= -1/den
        Jac [j+1][6]= x*(num_y)*np.power(den,-2)
        Jac [j+1][7]= y*(num_y)*np.power(den,-2)
        Jac [j+1][8]= num_y*np.power(den,-2)
#Calculates the error
    for i in range(0, len(f_x)):
        errx = (xrw[i]-f_x[i])
        erry = (yrw[i]-f_y[i])
        error.append(errx)
        error.append(erry)

    return error , Jac

#Function using openCV for the LM algorithm
def optim_H(H):
```

```
    h = np.array([H[0][0],H[0][1],H[0][2],H[1][0],H[1][1],H[1][2],H[2][0],H[2][1],H[2][2]])
    sol = optimize.root(func, h, jac = 'True', method='lm')
    H[0][0] = sol.x[0]
    H[0][1] = sol.x[1]
    H[0][2] = sol.x[2]
    H[1][0] = sol.x[3]
    H[1][1] = sol.x[4]
    H[1][2] = sol.x[5]
    H[2][0] = sol.x[6]
    H[2][1] = sol.x[7]
    H[2][2] = sol.x[8]
    return H
#Function to reproject corners using LM
def reproject_LM(H, img, wc, N, corner):
    new_cor = []
    for i in range (0, len(wc)):
        wc_new = np.matmul(np.linalg.inv(H),wc[i])
        wc_new = wc_new/wc_new[2]
        new_cor.append(wc_new)
        cv2.circle(img,(int(wc_new[0]),int(wc_new[1])), 2, (0,0,255), thickness = 1 )
        cv2.circle(img,(int(corner[i][0]),int(corner[i][1])), 2, (0,255,0), thickness = 1 )
    cv2.imwrite('C:/Users/namra/Desktop/Fall 2018/ECE 
661/hw8/reproject_LM/reproject_%i.jpg'%(N),img)
    E_dist = []
    for b in range(len(wc)):
        t = np.sqrt(np.square(corner[b][0]-new_cor[b][0]) + np.square(corner[b][1]-new_cor[b][1]))
        E_dist.append(t)
    mean = sum(E_dist)/len(E_dist)
    print ('Mean_LM_%i : %f'%(N,mean))
```

```
    var = sum(np.square(E_dist - mean))/len(E_dist)
    print('Variance_LM_%i : %f'%(N,var))



H = []
num = 1
corners = []

image_f = cv2.imread('C:/Users/namra/Desktop/Fall 2018/ECE 661/hw8/Files (1)/Dataset1/Pic_11.jpg')
h1, v1 = houghlines(image_f,100)
wc, img_fixed = createfixedcorners(image_f, h1, v1, 100)

cv2.imwrite('C:/Users/namra/Desktop/Fall 2018/ECE 661/hw8/fixed.jpg',img_fixed)

for i in range (1, 41):
    if i != 20 and i != 13 and i != 18 and num < 35 :
        image = cv2.imread('C:/Users/namra/Desktop/Fall 2018/ECE 661/hw8/Files (1)/Dataset1/Pic_%i.jpg'%(i))
        h_line, v_line = houghlines(image,i)
        num = num + 1
        corner,non = createcorners(image, h_line, v_line, i)
        corners.append(corner)
        H1 = homography(wc, corner)
        H.append(H1)
w = calc_w(H)
K = calc_k(w)



#j = 0
```

```
#i = 1

#cor = corners[j]

#P = calcRT(K, H[j])

#reproject(K, P,img_fixed, cor, i, wc)


#j = 0

#i = 1

#cor = corners[j]

#P = rod_P(K, H[j])

#h_lm = np.matmul(K,P)

#h_optim = optim_H(h_lm)

#reproject_LM(h_optim, img_fixed, corners[j], i , wc)
```