ECE 66100 HW8 Report

Zhengxin Jiang
(jiang839@purdue.edu)

November 15, 2022

1 Theory Question

No. We cannot see ω in the camera image because the absolute conic is on imaginary coordinates. Since the camera image of the absolute conic is independent of the rotation and translation matrices R and t, we can utilize $\omega = K^{-T}K^{-1}$ to find the intrinsic parameters.

2 Implementation

2.1 Corner detection

The first step of the whole implementation is to locate all the corner points in the calibration pattern. The step is done by following procedures:

Edge Detection

The Canny edge detector is used to extract the edges on the calibration pattern. In my implementation I used the cv2.Canny() function from OpenCV.

Lines fitting and filtering

Next we use Hough transform to fit lines onto the edges. The cv2.HoughLines() function is used. We sort the lines into vertical lines and horizontal lines for further intersection points extraction.

Once the function is called there might be several lines on the position where should be only one line. The line filtering is done by grouping the lines on the same position together and take the average.

Extracting Intersection Points

Once we have all the vertical lines and horizontal lines, we can extract intersection points by taking all the cross points of these lines.

2.2 Camera Calibration

Find Intrinsic Parameters

For calculating the intrinsic parameters, we start from the equations in lecture notes

$$\vec{h_1}^T \omega \vec{h_1} - \vec{h_2}^T \omega \vec{h_2} = 0$$

$$\vec{h_1}^T \omega \vec{h_2} = 0$$

We can convert them into the linear equation system

$$V\vec{b} = \begin{bmatrix} \vec{v_{12}^T} \\ (\vec{v_{11}} - \vec{v_{22}})^T \end{bmatrix} = 0$$

where

$$v_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix} \; and \; \vec{b} = \begin{bmatrix} \omega_{11} \\ \omega_{12} \\ \omega_{22} \\ \omega_{13} \\ \omega_{23} \\ \omega_{33} \end{bmatrix}$$

Once we have calculated ω, we can calculate the intrinsic matrix K by

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

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

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

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

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

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

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

Find Extrinsic Parameters

Start from the relationship

$$K^{-1}\begin{bmatrix} \vec{h_1} & \vec{h_2} & \vec{h_3} \end{bmatrix} = \begin{bmatrix} \vec{r_1} & \vec{r_2} & \vec{t} \end{bmatrix}$$

we derive that

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

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

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

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

Then we apply the scale factor $\xi = \frac{1}{||K^{-1}\vec{h_1}||}$ to R and t.

Finally we do the rotation matrix conditioning

$$Q = UDV^T$$

$$R = UV^T$$

Parameters Refinement

We can further do parameter refinement by applying the LM optimization.

3 Results

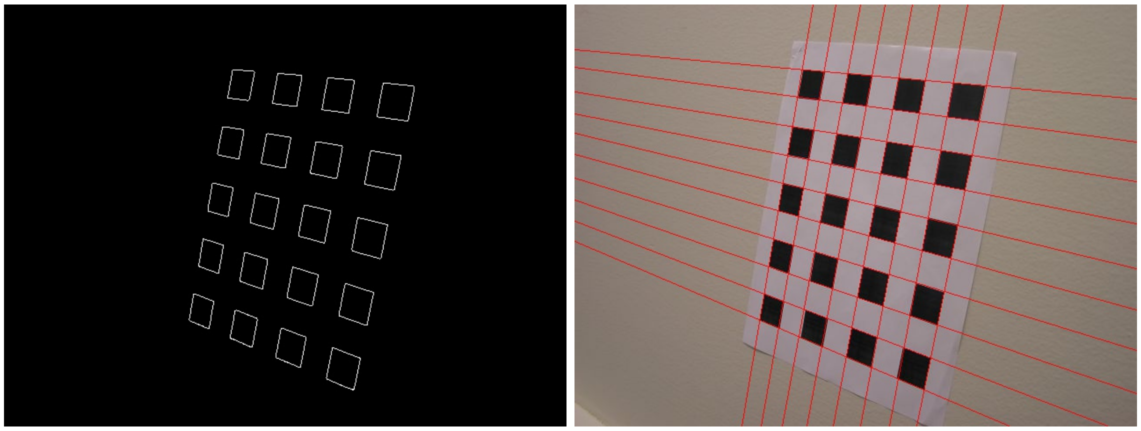

Figure 1: Edge detection and line fitting of image1

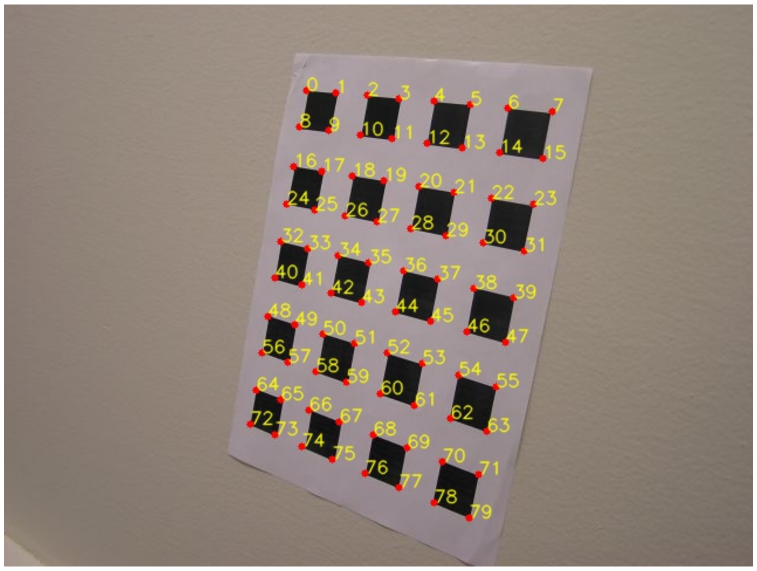

Figure 2: Corner points of image1

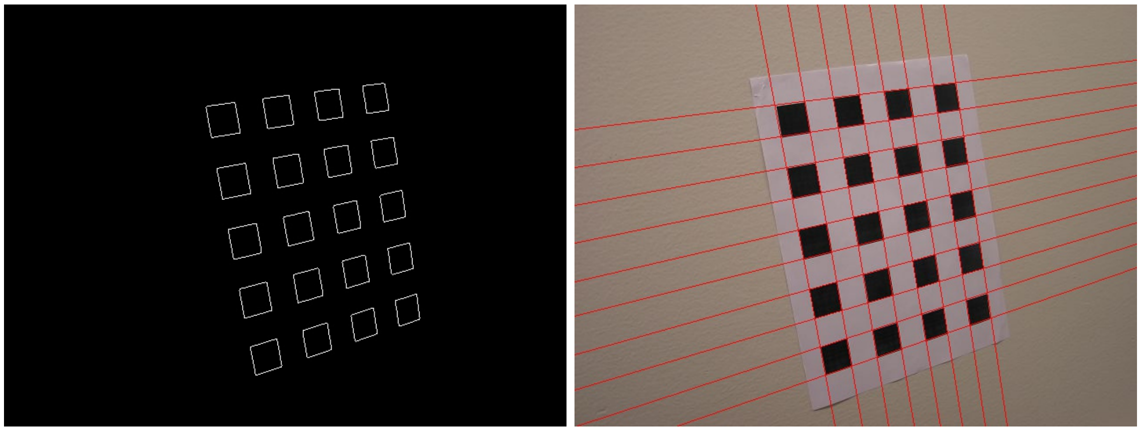

Figure 3: Edge detection and line fitting of image3

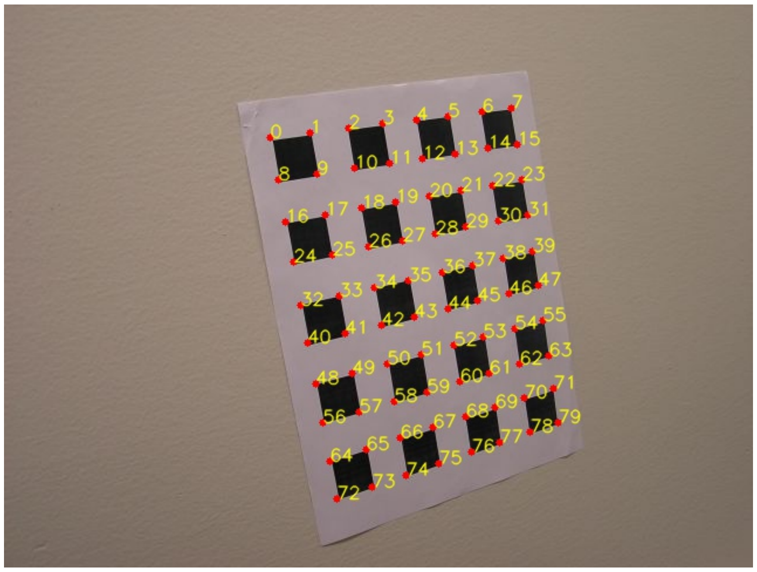

Figure 4: Corner points of image3

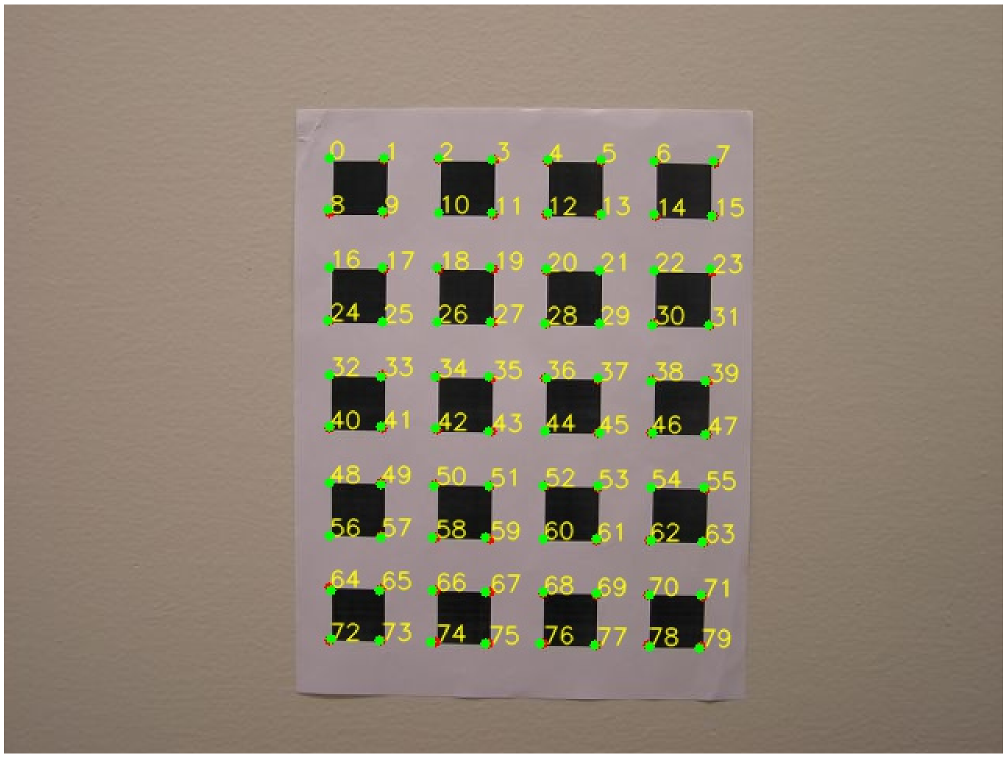

Figure 5: Reprojection of image1 to fixed image (image11) without LM

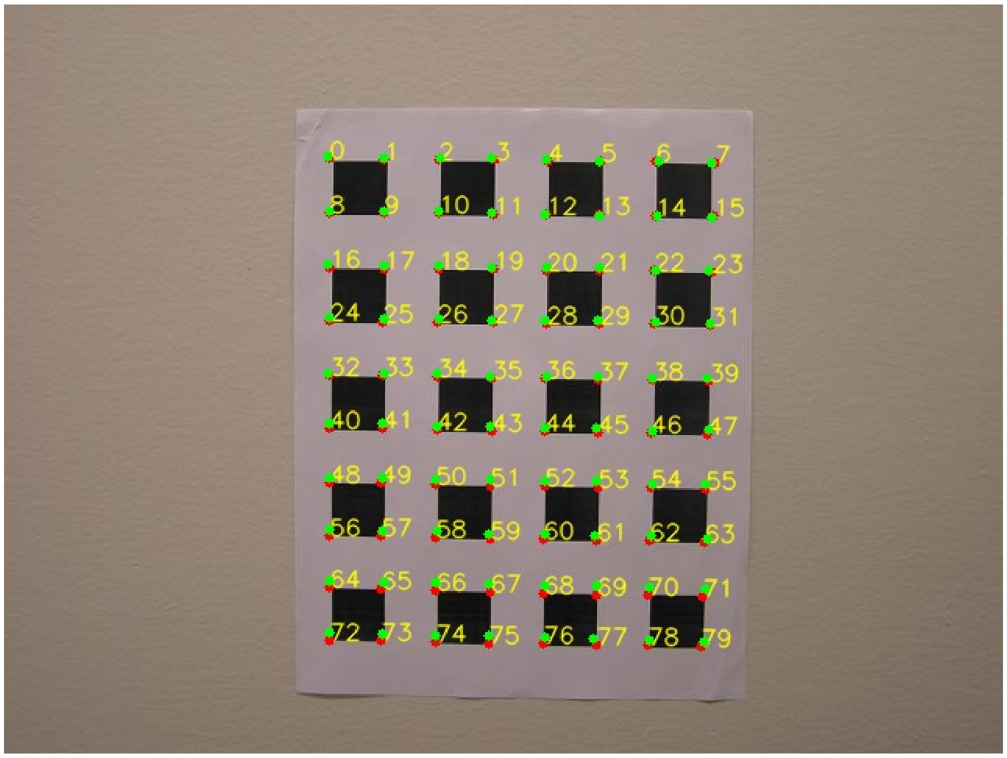

Figure 6: Reprojection of image3 to fixed image (image11) without LM

4 Source code

```
# ECE661 HW8
# Zhengxin Jiang
# jiang839

import cv2
import numpy as np
import matplotlib . pyplot as plt
import math
from scipy.optimize import least_squares


# load the whole image set
def loadImages(prefix, num):

    imgset = []

    for idx in range(1, num+1):
        img = cv2.imread('Dataset1/'+prefix+str(idx)+'.jpg')
        imgset.append(img)

    return imgset

def loadSelfImages(num):

    imgset = []

    for idx in range(num):
        img = cv2.imread('Dataset2/'+str(idx)+'.jpg')
        imgset.append(img)

    return imgset

# The function takes two arrays of points and return the homography from 1 to 2
def findHomograpy_LeastSquare(points_1, points_2):

    P = np.zeros((len(points_1)*2+1, 9))

    # construct a homogenious system
    for i in range(len(points_1)):

        P[2*i, 0] = -points_1[i][0]
        P[2*i, 1] = -points_1[i][1]
        P[2*i, 2] = -1
        P[2*i, 6] = points_1[i][0]*points_2[i][0]
        P[2*i, 7] = points_1[i][1]*points_2[i][0]
        P[2*i, 8] = points_2[i][0]

        P[2*i+1, 3] = -points_1[i][0]
        P[2*i+1, 4] = -points_1[i][1]
        P[2*i+1, 5] = -1
        P[2*i+1, 6] = points_1[i][0]*points_2[i][1]
        P[2*i+1, 7] = points_1[i][1]*points_2[i][1]
        P[2*i+1, 8] = points_2[i][1]

    P[-1, -1] = 1

    b = np.zeros(len(points_1)*2+1)
    b[-1] = 1

    # solving H using least square method
    H = np.linalg.lstsq(P, b, rcond=None)[0]
    H = H/H[-1]
    H = np.reshape(H, (3,3))

    return H


# given a rho-sorted line set, return the filtered lines
def lineFilter(lines, thresh):

    lines_filtered = []
    group = []

    for i in range(len(lines)):

        if i == 0:
            group.append(lines[i])
        elif (abs(lines[i][0] - lines[i-1][0])) < thresh:
            group.append(lines[i])
        else:
            lines_filtered.append(np.mean(group, axis=0))
            group = []
            group.append(lines[i])

    lines_filtered.append(np.mean(group, axis=0))

    return np.array(lines_filtered)


# convert lines to points on the line
def lineToPoint(lines):

    p1, p2 = [], []

    for rho,theta in lines:

        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 - 1000*(-b))
        y1 = int(y0 - 1000*(a))
        x2 = int(x0 + 1000*(-b))
        y2 = int(y0 + 1000*(a))

        p1.append((x1, y1))
        p2.append((x2, y2))

    return np.array(p1), np.array(p2)


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

    intersects = []

    vp1 = np.append(vp1, np.ones((len(vp1),1)), axis=1)
    vp2 = np.append(vp2, np.ones((len(vp2),1)), axis=1)
    hp1 = np.append(hp1, np.ones((len(hp1),1)), axis=1)
    hp2 = np.append(hp2, np.ones((len(hp2),1)), axis=1)

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

    for hline in hlines:

        its_temp = np.cross(vlines, hline)
        its_temp = (its_temp.T/its_temp.T[-1]).T.astype(int)

        intersects.append(its_temp)

    intersects = np.concatenate(intersects,axis=0)

    return intersects


# find the intersection points in an image
def intersectionsInImage(image, outputLinePoints=False):

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

    # extract edges and lines
    edges = cv2.Canny(img,255,400)
    lines = np.squeeze(cv2.HoughLines(edges, 1, np.pi/180, 50))

    # line filter
    vlines = lines[np.where(abs(lines[:,1]-np.pi/2)>(np.pi/4) )]
    for i in range(len(vlines)): # dealing with nearly vertical/horizontal lines
        if vlines[i][0]<0 and abs(vlines[i][1]-np.pi)<0.2:
            vlines[i][0] = abs(vlines[i][0])
            vlines[i][1] -= np.pi
    vlines = np.array(sorted(vlines, key=lambda x: x[0]))
    vlines = lineFilter(vlines, 10)
    if (vlines[:,0]<0).all(): # maintain the correct intersection points order
        vlines = np.array(sorted(vlines, key=lambda x: abs(x[0])))

    hlines = lines[np.where(abs(lines[:,1]-np.pi/2)<(np.pi/4) )]
    for i in range(len(hlines)):
        if hlines[i][0]<0 and abs(hlines[i][1]-np.pi)<0.2:
            hlines[i][0] = abs(hlines[i][0])
            hlines[i][1] -= np.pi
    hlines = np.array(sorted(hlines, key=lambda x: x[0]))
    hlines = lineFilter(hlines, 10)
    if (hlines[:,0]<0).all():
        hlines = np.array(sorted(hlines, key=lambda x: abs(x[0])))

    vp1, vp2 = lineToPoint(vlines)
    hp1, hp2 = lineToPoint(hlines)

    # calculate intersections
    intersects = findIntersections(vp1, vp2, hp1, hp2)

    if outputLinePoints:
        return intersects, vp1, vp2, hp1, hp2
    else:
        return intersects


# Find the intrinsic and extrinsic parameters of an image set
def findParameters(imgset, world_coords):

    # caluculate the per-image homographies
    H_list = []
    validimg_indices = []
    intersects_list = []

    for i in range(len(imgset)):
        intersects = intersectionsInImage(imgset[i])

        if len(intersects) == 80:
            H = findHomograpy_LeastSquare(world_coords, intersects)
            H_list.append(H)
            validimg_indices.append(i)
            intersects_list.append(intersects)

    # calculate omega
    num = len(H_list)
    V = np.zeros((num*2+1, 6))

    for i in range(num):

        H = H_list[i]

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

    V[-1, -1] = 1

    b = np.zeros(num*2+1)
    b[-1] = 1

    omega = np.linalg.lstsq(V, b, rcond=None)[0]

    # calculate intrinsic parameters K
    y0 = (omega[1]*omega[3]-omega[0]*omega[4])/(omega[0]*omega[2]-omega[1]**2)
    lamda = omega[5]-(omega[3]**2+y0*(omega[1]*omega[3]-omega[0]*omega[4]))/omega[0]
    ax = np.sqrt(lamda/omega[0])
    ay = np.sqrt(lamda*omega[0]/(omega[0]*omega[2]-omega[1]**2))
    s = -omega[1]*ax**2*ay/lamda
    x0 = s*y0/ay - omega[3]*ax**2/lamda

    K = np.array([[ax, s, x0],[0, ay, y0],[0, 0, 1]])

    # calculate extrinsic parameters R and t
    R_list=[]
    t_list=[]

    for H in H_list:

        # K inverse multiply h, then apply scale factor
        temp = np.dot(np.linalg.inv(K),H)
        temp = temp/np.linalg.norm(temp[:,0])

        r3 = np.cross(temp[:,0], temp[:,1])
        t = temp[:,2].copy()

        temp[:,2] = r3

        # conditioning
        u, _, v = np.linalg.svd(temp)
        R = np.dot(u, v)

        R_list.append(R)
        t_list.append(t)

    return K, R_list, t_list, np.array(validimg_indices), intersects_list



# Do repeojection from a source image to destination image
def reprojection(source_img_num, dest_img_num, intersects_list, H_rep_list,
    validimg_indices):

    idx_s = np.where(validimg_indices==source_img_num-1)[0]
    idx_d = np.where(validimg_indices==dest_img_num-1)[0]

    if len(idx_s) == 1 and len(idx_d) == 1:

        spoints = intersects_list[idx_s[0]]

        Hs = H_rep_list[idx_s[0]]
        Hd = H_rep_list[idx_d[0]]
        H_sd = np.dot(Hd, np.linalg.inv(Hs))


    dpoints = np.dot(H_sd, spoints.T)
    dpoints = np.divide(dpoints, dpoints[-1]).T

    return dpoints.astype(int)


if __name__ == '__main__':

    result_path = 'C:/Users/jzx/OneDrive␣-␣purdue.edu/ECE661/hw8/result␣images/'

    # load image set
    imgset = loadImages('Pic_', 40)

    # image set with intersects labeled
    imgset_labeled = []

    for i in range(len(imgset)):

        img = imgset[i].copy()

        intersects = intersectionsInImage(img)
        intersects = intersects[:, :2]

        for i in range(len(intersects)):
            cv2.circle(img,tuple(intersects[i]),radius=3,color=(0,0,255),thickness=-1)
            cv2.putText(img,str(i),tuple(intersects[i]),cv2.FONT_HERSHEY_SIMPLEX,0.5,
                (0,255,255),1,cv2.LINE_AA)

        imgset_labeled.append(img)

    # create world coordinates for corner points
    h = np.linspace(0, 350, 8)
    v = np.linspace(0, 450, 10)

    world_coords = np.zeros((len(v)*len(h), 2))
    for i in range(len(world_coords)):
        world_coords[i] = [h[i%8], v[i//8]]

    # calculate parameters
    K, R_list, t_list, validimg_indices, intersects_list = findParameters(imgset,
        world_coords)

    # calculate back project homographies from world coordinates to images
    H_back_list = []

    for i in range(len(R_list)):

        imgidx = validimg_indices[i]

        temp = R_list[i]
        temp[:,2] = t_list[i]
        H_back = np.dot(K, temp)
        H_back_list.append(H_back)

    # reproject to the fixed image
    fixed_img = 11
    rep_img = 1

    rep_points = reprojection(rep_img, fixed_img, intersects_list, H_back_list,
        validimg_indices)

    img_rep = imgset_labeled[fixed_img-1].copy()

    for i in rep_points:
        cv2.circle(img_rep,tuple(i[:2]),radius=3,color=(0,255,0),thickness=-1)

    cv2.imwrite(result_path + 'rep_1to11' + '.jpg', img_rep)

    img_rep = cv2.cvtColor(img_rep, cv2.COLOR_BGR2RGB)
    plt.figure()
    plt.imshow(img_rep)
```