CS 512 HW4 REPORT

SUDIPTA SWARNAKAR

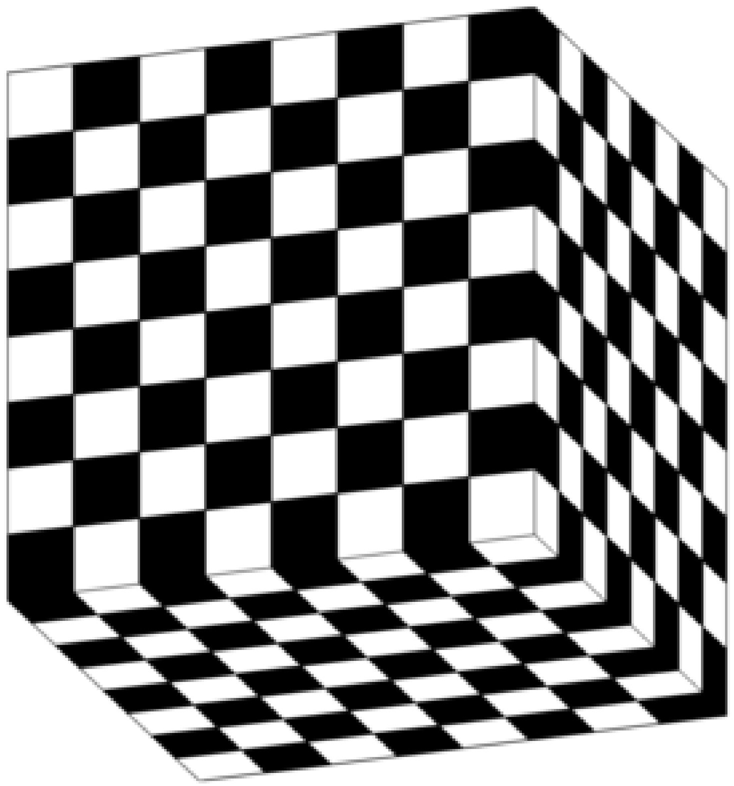
CWID: A20377210

Problem Statement 1: To extract non-planar feature points from calibration target (3D Cube) and show them on image.

Sol: I used the OpenCV cv2.goodFeaturestoTrack() method to get the
corners of the 3D Cube image. Here we need to mention how many corners we need to identify (initialized it with 70 points). cv2.goodFeaturestoTrack() returns those good features points from that 3D Cube.

i) World Points (3D points)

ii) Image Points (2D points)



Input 3D Image

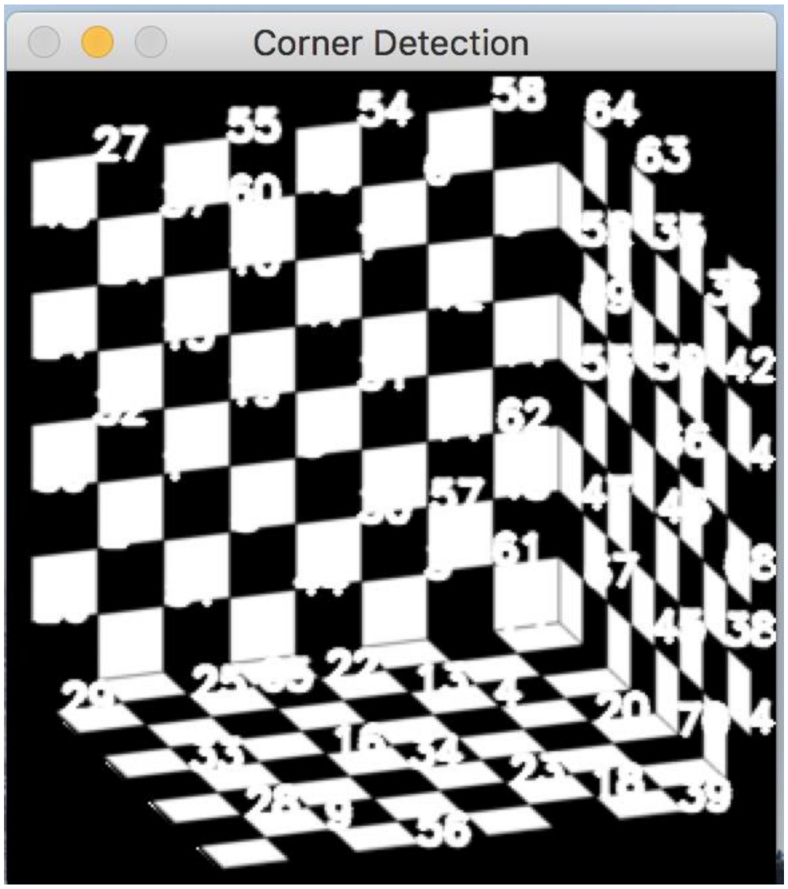


Image with Features Points Detected

Problem Statement 2:
How to get the world points for the corresponding image points.

Sol: From the above method we also get 3D world points and 2D image points as output files. Here while generating world points I am assuming that the camera is not moving along Z axis so I am considering Z=0 while finding world points.

Problem Statement 3:
Implement Camera Calibration Algorithm.

Sol:
Here my code is taking merged world points and image points as one .csv file to implement non-planar calibration algorithm for camera. This method is calculating a new matrix based on the below equation, Ax=0

For points 1,first two rows of matrix A are calculated as follows-
Row1 = [X_,Y_,Z_1,0,0,0,0,-x*X_,-x*Y_,-x*Z_,-x]

Row2 = [0,0,0,0, X_,Y_,Z_1, -y*X_,-y*Y_,-y*Z_,-y]
After that I am calculating this matrix for all given n points. This matrix is of the size 2n*12 and to get the 3*4 projection matrix, we take the last column of V of SVD(A). At each step after getting matrix M, we need to scale it by S factor to get the values. To find intrinsic and extrinsic parameters my getcameraParams() method is taking M,b, 2D Image points and 3D world points.

Source Code

```
M = np.array(M)
M.reshape(3,4)
m1 = np.array(M[0])
m2 = np.array(M[1])
m3 = np.array(M[2])
actualPoint2DList = []

#Finding Actual Points and Mean Square Error
count = 0
mse = 0
for points in pointList3D:
    actualPoint2D = []
    points.append(1)
    points = np.array(points)
    xpoint2D = float(np.dot((m1.T),points))/float(np.dot((m3.T),points))
    ypoint2D = float(np.dot((m2.T),points))/float(np.dot((m3.T),points))
    actualPoint2D.append((xpoint2D,ypoint2D))
    actualPoint2DList.append(actualPoint2D)

    mse += pow((pointList2D[count][0] - xpoint2D ),2) + pow((pointList2D[count][1] - ypoint2D),2)
    count +=1

mse = mse/len(pointList2D)

a1 = (m1[:3]).T
a2 = (m2[:3]).T
a3 = (m3[:3]).T

#Solving for equation parameters
z = (a3 * a3).sum()
pDet = 1 / (z ** 0.5)

print "Pdeter",pDet

u0 = float(np.dot((pDet * pDet), np.dot(a1,a3)))
u0 = round(u0,2)

v0 = float((pDet * pDet) * np.dot(a2, a3))
v0 = round(v0, 2)

print "u0,v0",u0,v0

alphav = math.sqrt(float(np.dot(pow(pDet ,2), np.dot(a2,a2)) - v0*v0))
alphav = round(alphav,2)

s = (float(alphav* np.dot(np.cross(a1,a3), np.cross(a2,a3))))
```

```
s = math.ceil(s)

alphau = (float(pow(pDet, 2)*np.dot(a2,a2) - pow(s,2) - pow(v0, 2)) ** 0.5)
alphau = round(alphau,2)

print "alphau,alphav",alphau,alphau

K = np.array([[alphau, s, u0],[0,alphav,v0],[0,0,1]]).reshape(3,3)

print "K",K

e = np.sign(b[2])

inv = np.linalg.inv(K)
T = e * pDet * np.dot(inv,b)

r3= e * pDet* a3
r1 = np.cross((alphav*a2),a3)
r2= np.cross(r1,r3)
R = [r1,r2,r3]
R= np.array(R).reshape(3,3)
```

Output on Given Test Data:

```
Non planar camera calibration
Estimated Parameters:
72
Pdeter 41674.3802654
u0,v0 173.76 190.37
alphau,alphav 10.58 10.58
K [[  10.58   -0.    173.76]
 [   0.     10.58  190.37]
 [   0.      0.      1.  ]]

('Mean Square Error', 16387361.261390587)
('Intrinsic Parameter- \nK:', array([[  10.58,   -0.  ,  173.76],
       [   0.  ,   10.58,  190.37],
       [   0.  ,    0.  ,    1.  ]]))
('Extrinsic Parameters - \nT:', array([-274.09086815, -265.75282081,  177.32192177]), '\nR:', array([[  3.15418365e-08,
-4.82175508e-08,   2.93622913e-08],
       [  4.57494353e-08,   2.12510134e-09,  -4.56556341e-08],
       [ -5.11483387e-01,  -6.65565490e-01,  -5.43513867e-01]]))
```

Problem Statement 4:

Implement RANSAC Algorithm

Sol:

In order get Robust estimate using RANSAC algorithm, I am taking into consideration the noise data given by Prof. My RANSAC method takes 2D image points, 3D world point, number of trials and number of points to be drawn. My RANSAC implementation is using all necessary parameters from given RANSAC.config file. My method doesn't exceed the k_max, each time count is increased.

Source Code:

```
p = 0.99 #Given
k = math.ceil(np.log(1-0.99)/np.log(1-math.pow(0.5,n))) #Calculating number of trials

while(count<k):
    count = count +1
    if (count>k_max):
        break
    else:
        test = random.sample(pointList,n)
        mat, worldpoint, imagepoint = getMatrix(test)

        projection_matrix = projection(mat)
        threshold = getThreshold(projection_matrix, test)

        new_s, length = getInliers(projection_matrix, pointList, threshold)
        w = (len(new_s)/float(len(pointList)))
        k = math.ceil(np.log(1-0.99)/np.log(1-math.pow(w,n)))

        if(length> 6):
            all_S.append(new_s)
            all_len.append(length)

b = totalset[np.argmax(l)]
matt, worldpoint, imagepoint = getMatrix(b)
Matrix_M = projection(matt)
b1 = []
for values in Matrix_M:
    b1.append(values[3])
return Matrix_M, b1
```

Method getMatrix(data) returns last column of V of SVD of matrix returned from getMatrix() and it can be given as,

```
data = np.array(data)
finalmatrix = []
wpoint = []
ipoint = []
for i in data:
    X_,Y_,Z_,x,y = i.split(' ')
    X_ = float(X_)
    Y_ = float(Y_)
    Z_ = float(Z_)
    x = float(x)
    y = float(y)

    a = X_,Y_,Z_,1,0,0,0,0,-x*X_,-x*Y_,-x*Z_,-x
    b = 0,0,0,0,X_,Y_,Z_,1,-y*X_,-y*Y_,-y*Z_,-y
    c = X_,Y_,Z_,1
    d= x,y
    temp =[]
    temp.extend(a)
    finalmatrix.append(temp)
    temp = []
    temp.extend(b)
    finalmatrix.append(temp)
    wpoint.append(c)
    ipoint.append(d)
```

Method for Finding Inliers :

```
a, wPoint, iPoint = getMatrix(data)

inlierP = []
for i in range(len(data)):
    ui,vi,wi=np.matmul(M, wPoint[i])
    if not wi == 0:
        u = float(ui)/float(wi)
        v=float(vi)/float(wi)
    else:
        u = ui
        v = vi
    a,b = iPoint[i]
    dist = math.sqrt(math.pow(a-u,2) + math.pow(b-v,2))

    if(dist < thres):
        inlierP.append(data[i])
return inlierP, len(inlierP)
```