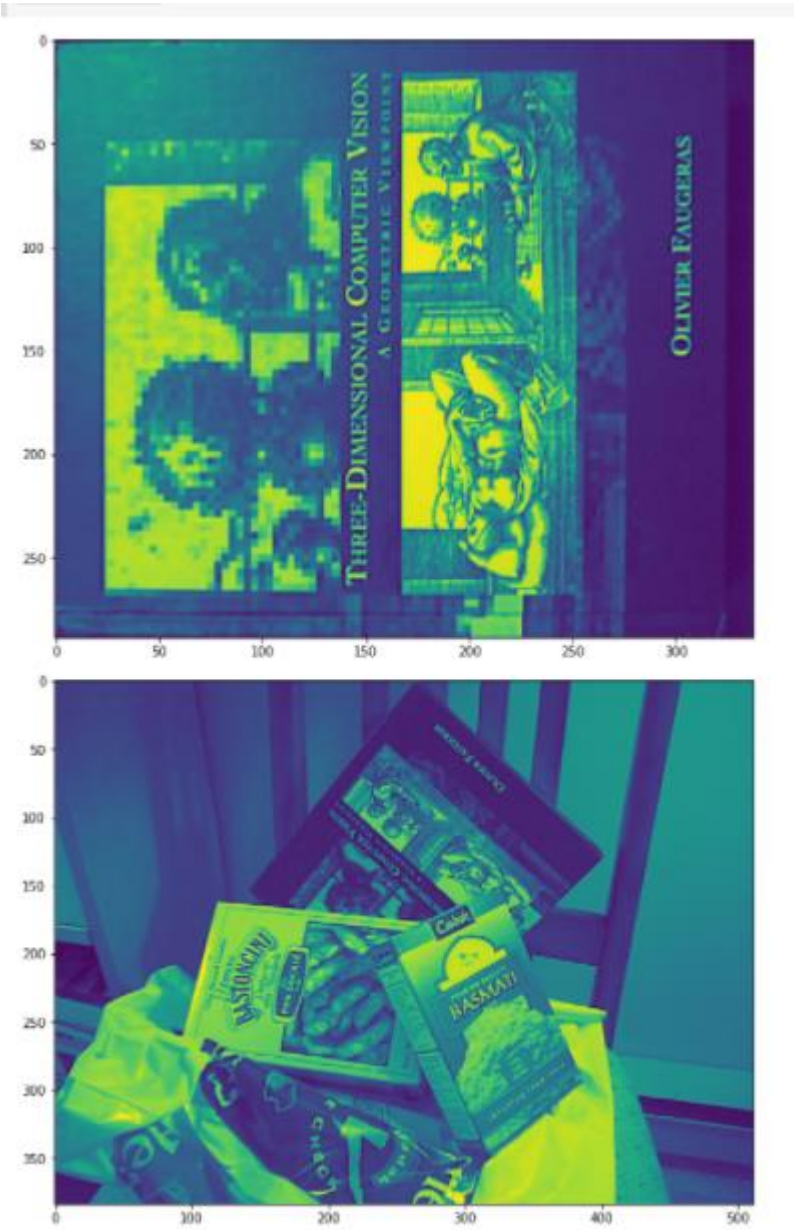
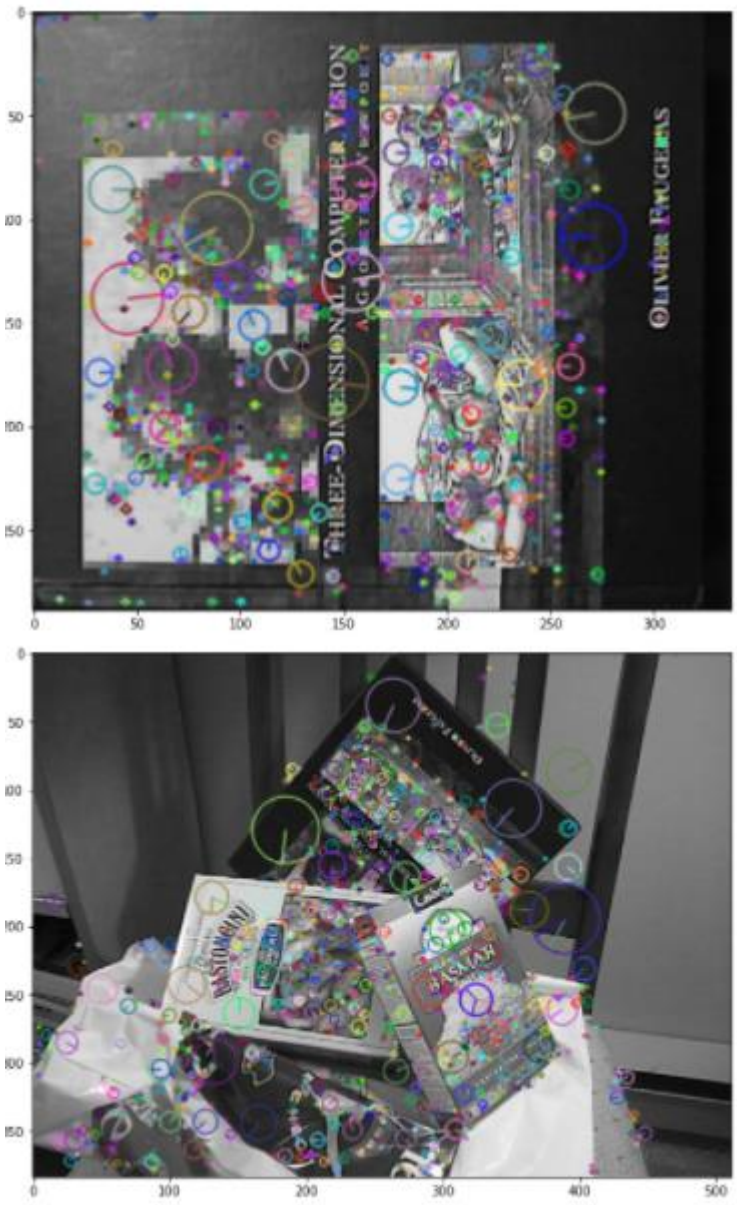
```
#!pip install pysift
#To grader: If you want to run this on colab, make sure to get py
sift.py uploaded.
from scipy import linalg
import random
import cv2 as cv
import pysift
import numpy as np
import scipy
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import *


images = ['book.pgm', 'scene.pgm']
image_book = cv.imread(images[0], 0)
image_scene = cv.imread(images[1], 0)

kpb, deb = pysift.computeKeypointsAndDescriptors(image_book)
kps, des = pysift.computeKeypointsAndDescriptors(image_scene)

img_dispb=cv.drawKeypoints(image_book,kpb, None, flags=cv.DRAW_MA
TCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img_disps=cv.drawKeypoints(image_scene,kps, None, flags=cv.DRAW_M
ATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

def plot(image):
    plt.figure(figsize = (10,10))
    plt.imshow(image)

plot(image_book)
plot(image_scene)
```

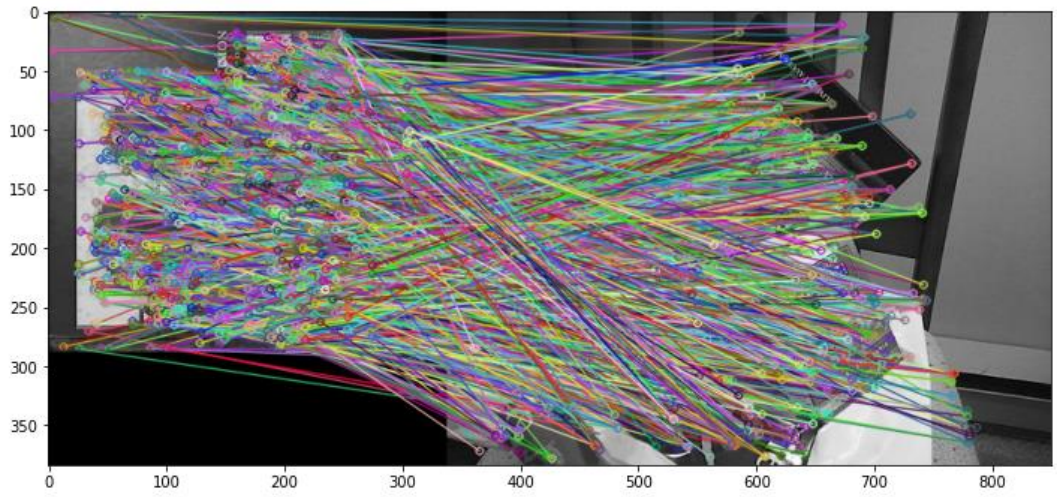



```
matchlines = []
bf = cv.BFMatcher()
matches = bf.knnMatch(deb, des, k=2)

threshold = 0.8
#append it to
for x,y in matches:
    if x.distance < threshold *y.distance:
        matchlines.append(x)

matches = sorted(matchlines, key =lambda x:x.distance)
img = image_book
img = cv.drawMatches(image_book,kpb,image_scene,kps,matches[:10],
img,flags=2)
```

```
book_plot = cv.drawKeypoints(image_book, kpb, outImage = None, fl
ags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plot(book_plot)
scene_plot = cv.drawKeypoints(image_scene,kps,outImage = None,fla
gs=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plot(scene_plot)
```



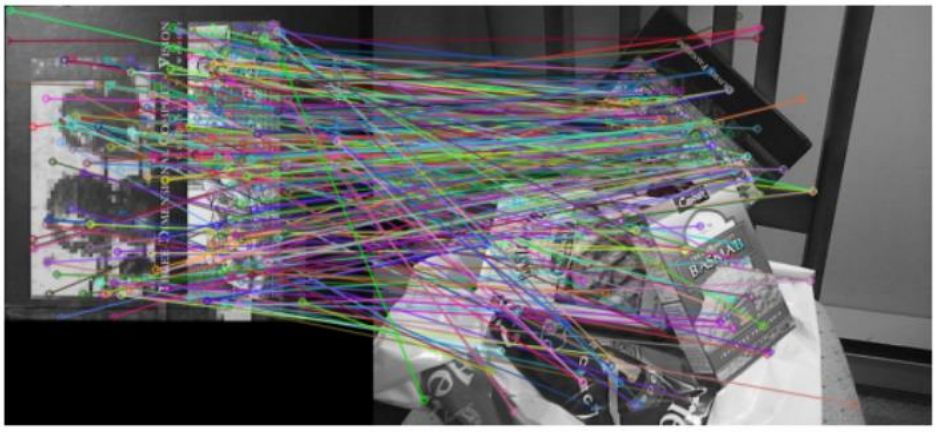
```
def Matchline(img_disp1, kp1, des1, img_disp2, kp2, des2):
        matches = bf.knnMatch(queryDescriptors=des1,trainDescriptors=des2, k=2)
        img3 = cv.drawMatchesKnn(img_disp1,kp1,img_disp2,kp2,matches,None,flags=2)
        plt.figure(figsize=(12,15))
        plt.imshow(img3)
        plt.show()

Matchline(image_book,kpb,deb,image_scene, kps, des)
plt.axis('off')
```



(0.0, 1.0, 0.0, 1.0)

```
matches = bf.knnMatch(queryDescriptors=deb,trainDescriptors=des, k=2)
good = []
good_without_list = []
for m,n in matches:
        if m.distance < 0.9*n.distance:
                good.append([m])
                good_without_list.append(m)
matches_img = cv.drawMatchesKnn(image_book,kpb,image_scene,kps,good,flags=2,outImg=None)
plot(matches_img)
plt.axis('off')
```

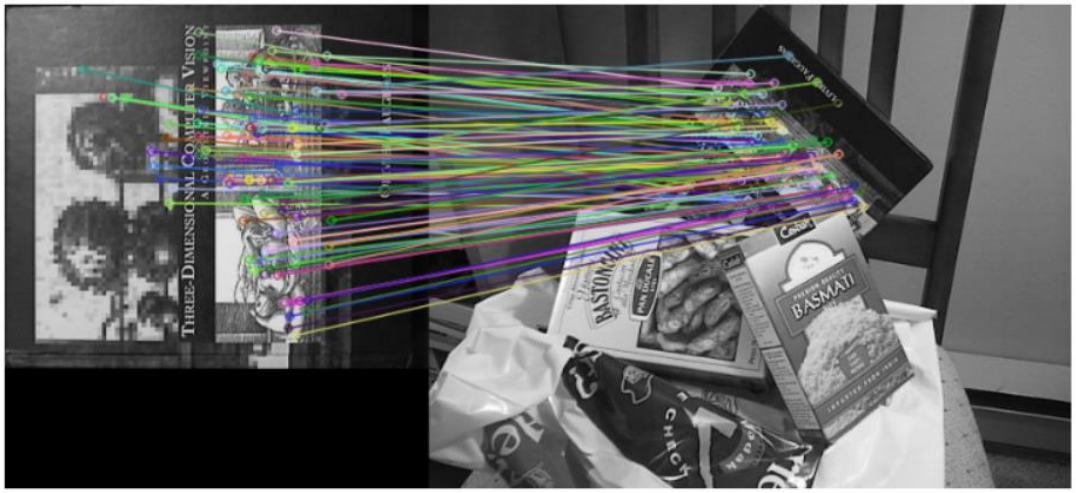
(-0.5, 849.5, 383.5, -0.5)



```
src_pts = np.float32([kpb[m.queryIdx].pt for m in good_without_li
st])
dst_pts = np.float32([kps[m.trainIdx].pt for m in good_without_li
st])

N = 100

max_count = 0
best_inliers = 0
best_q = 0
for i in range(N):
    pick_matches = np.random.choice(range(len(src_pts)), 3)
    A = np.zeros([6,6])
    B = np.zeros([6])
    for j in range(3):
        A[2*j][0] = src_pts[pick_matches[j]][0]
        A[2*j][1] = src_pts[pick_matches[j]][1]
        A[2*j][2] = 1
        A[2*j+1][3] = src_pts[pick_matches[j]][0]
        A[2*j+1][4] = src_pts[pick_matches[j]][1]
        A[2*j+1][5] = 1
        B[2*j] = dst_pts[pick_matches[j]][0]
        B[2*j+1] = dst_pts[pick_matches[j]][1]
    try:
        q = np.linalg.solve(A, B)
    except:
        continue


    count = 0
    inliers = []
    for j in range(len(src_pts)):
        transform = np.dot(np.array([[src_pts[j][0],src_pts[j][1]
,1,0,0,0],[0,0,0,src_pts[j][0],src_pts[j][1],1]]),q)
        if np.linalg.norm(transform - dst_pts[j]) < 10:
            count += 1
            inliers.append(good_without_list[j])
    if count>max_count:
        max_count = count
        best_inliers = inliers
        best_q = q
print('max count:',max_count)
```

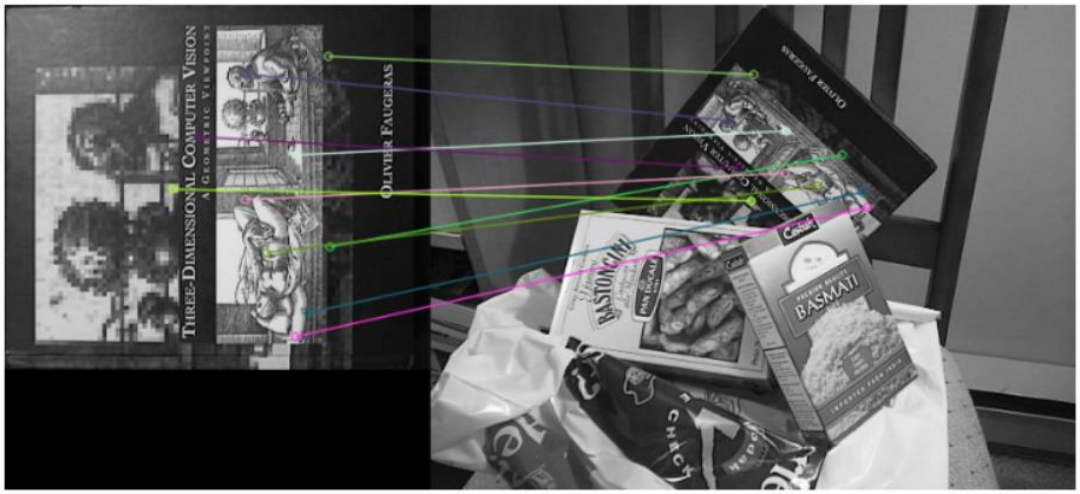
```
matches_img = cv.drawMatches(image_book,kpb,image_scene,kps,best_
inliers,flags=2,outImg=None)
plt.figure(figsize = (13,13))
plt.imshow(matches_img,cmap='gist_gray');
plt.axis('off');
```

```
max count: 114
```




```
fig = plt.figure()
plt.figure(figsize = (13,13))
plt.axis('off')
plt.imshow(img)
plt.plot()
```

```
[]
<Figure size 432x288 with 0 Axes>
```




```
H_matrix = np.zeros([3,3])
H_matrix[0][0] = best_q[0]
H_matrix[0][1] = best_q[1]
H_matrix[0][2] = best_q[4]
H_matrix[1][0] = best_q[2]
H_matrix[1][1] = best_q[3]
H_matrix[1][2] = best_q[5]
```

```
H_matrix[2][2] = 1
H_matrix = best_q.reshape(2,3)

pose = cv.warpAffine(image_book, H_matrix, (image_scene.shape[1],
 image_scene.shape[0]))
#plt.imshow(pose)

fig, axs = plt.subplots(1, 2, figsize=(15, 8))
axs[1].imshow(pose)
axs[1].axis('off')
axs[0].imshow(image_scene,cmap='gist_gray')
axs[0].axis('off')
```

```
(-0.5, 511.5, 383.5, -0.5)
```

```
def Hmatrix(H):
    return np.append(H, np.ones((H.shape[0],H.shape[1],1)),axis=2
)


#process data
candidates = np.array(match_candidates)
src = np.array([kpb[match.queryIdx].pt for match in candidates])
src = src.reshape(-1,1,2)
src = Hmatrix(src)
#dataset
dst = np.array([kps[match.trainIdx].pt for match in candidates])
dst = dst.reshape(-1,1,2)
dst = Hmatrix(dst)
data = []
for i in range(len(src)):
    data.append([src[i],dst[i]])
best_cnt = 0
sz = len(data)

#inliers
for round in range(100):
```

```
    inliers = []

    idx = random.sample(data,3)
    A = np.zeros((1,6))
    tvec = np.zeros((1,1))
    for i in range(3):
        A = np.append(A, np.append(idx[i][0],np.zeros((1,3)),axis
 =1),axis = 0)
        A = np.append(A, np.append(np.zeros((1,3)),idx[i][0],axis
=1),axis = 0)
        tvec = np.append(tvec, idx[i][1].reshape((3,1))[:-
1],axis=0)
    A = A[1:,:]
    tvec = tvec[1:,:]
    try:
        solution = np.linalg.solve(A,tvec)
    except:
        continue
    for i in range(sz):
        x = solution.reshape((2,3)).dot(data[i][0].T).reshape((1,
2))
        y = np.delete(data[i][1],2,axis=1).reshape((1,2))
        if np.linalg.norm(x-y)<10:
            inliers.append(i)
    if len(inliers) > best_cnt:
        best_inliers = inliers
        best_cnt = len(inliers)
        best_solution = solution

print('inliers:\n', best_cnt)
best_solution = best_solution.reshape((2,3))
print('RANSAC value is:\n', best_solution)
homography, mask = cv.findHomography(src,dst,cv.RANSAC,ransacRepr
ojThreshold =2.0)
print('Homography Matrix:\n',homography)


pose = cv.warpAffine(image_book, best_solution.reshape((2,3)),(im
age_scene.shape[1], image_scene.shape[0]))

sz = len(best_inliers)
A = np.zeros((1,6))
tvec = np.zeros((1,1))
idx = data
```

```
for p in range(sz):
    i = best_inliers[p]
    A = np.append(A, np.append(idx[i][0], np.zeros((1,3)), axis=1
), axis = 0)
    A = np.append(A, np.append(np.zeros((1,3)), idx[i][0], axis =
1), axis = 0)
    tvec = np.append(tvec, idx[i][1].reshape((3,1))[:-1],axis=0)
A = A[1:, :]
tvec = tvec[1:, :]
refit_solution = (np.linalg.lstsq(A,tvec,rcond=None)[0])

print('refit_solution:\n', refit_solution.reshape((2,3)))
pose = cv.warpAffine(image_book, refit_solution.reshape((2,3)),(i
mage_scene.shape[1], image_scene.shape[0]))

inliers:
 109
RANSAC value is:
 [[  0.38418662   0.4523256  138.77915484]
 [ -0.48327633   0.41748411 161.7178536 ]]
Homography Matrix:
 [[ 2.90240357e-01  4.45145063e-01  1.41707637e+02]
 [-4.66800005e-01  3.97871861e-01  1.53512503e+02]
 [-3.42747238e-04  7.23351591e-05  1.00000000e+00]]
refit_solution:
 [[  0.40077908   0.4522761  135.13846632]
 [ -0.45571729   0.41782549 155.4069019 ]]
```

```
import numpy as np
from scipy import linalg

world = np.loadtxt("world.txt")
image = np.loadtxt("image.txt")
# lenw = len(world)
# leni = len(image)
image_vector = np.concatenate((image,np.ones([1,image.shape[1]]))
,axis=0)
world_vector = np.concatenate((world,np.ones([1,world.shape[1]]))
,axis=0)

#Get matrix M
M = np.zeros([2*image_vector.shape[1],12])
for i in range(image_vector.shape[1]):
    M[i][4:8] = -image_vector[2][i] * world_vector[:,i]
    M[i][8:12] = image_vector[1][i] * world_vector[:,i]
    M[10+i][:4] = image_vector[2][i]*world_vector[:,i]
    M[10+i][8:12] = -image_vector[0][i]*world_vector[:,i]
#奇异值进行分解U(M,M)S(M.N)V(N,N)
U,S,V = np.linalg.svd(M)
P = V[-1:].reshape(3,4)

projection2 = np.dot(P,world_vector)
projection22 = projection2/projection2[-1]

print('Camera Intrinsic Matrix:')
print(np.round_(projection22,3))
np.round_(image_vector,3)
```

```
Camera Intrinsic Matrix:
[[ 5.118  5.524  7.163  5.222  5.605 13.595  8.735  6.224  9.748  5.09 ]
 [ 4.765  3.87   7.359  4.428  4.675 10.052  5.564  3.908  6.904  4.551]
 [ 1.     1.     1.     1.     1.     1.     1.     1.     1.     1.   ]]
array([[ 5.118,  5.524,  7.163,  5.222,  5.605, 13.595,  8.735,  6.224,
         9.748,  5.09 ],
       [ 4.765,  3.87 ,  7.359,  4.428,  4.675, 10.052,  5.564,  3.908,
         6.904,  4.551],
       [ 1.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,
         1.   ,  1.   ]])
```

```
#奇异值进行分解U(M,M)S(M.N)V(N,N)
U, S, V = np.linalg.svd(P)
C = V[-1]
C = C[:-1]/C[-1]
print('Project center is :',C)
```

```
Project center is : [ 1. -1. -1.]
```

```
r,q = linalg.rq(P, mode='economic')
R = (q.T)[:-1].T
t = (q.T)[-1].T
C_est = np.linalg.solve(-R,t)
print('C estimation is:',C_est)
# if(C_est.any == C.any):
print('Aanswer is the same.')
```

```
C estimation is: [ 1. -1. -1.]
Aanswer is the same.
```

```
import numpy as np
import scipy.io as sio
import pylab
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

sfm = sio.loadmat('sfm_points.mat')
center = np.zeros((2,1))
W = np.zeros((20, 600))

for i in range(10):
    tmp = sfm["image_points"][:, :, i]
    len = tmp.shape[1]
    x = np.sum(tmp[0, :])/len
    y = np.sum(tmp[1, :])/len
    points = np.array([x,y])
    center = np.append(center, points.reshape(-1,1), axis=1)

center = center[:, 1:]


for i in range(10):
    tmp = sfm["image_points"][:, :, i] - center[:, i].reshape(2,1
)
    W[i] = tmp[0, :]
    W[i+10] = tmp[1, :]
#print(W)
print ('t_i (first camera): \n', center[:,0])
print ('Center for each camera t_i: \n',center[:,0:])
```

```
t_i (first camera):
 [2.36847579e-17 8.28966525e-17]
Center for each camera t_i:
 [[ 2.36847579e-17 -3.55271368e-17  9.47390314e-17  3.07901852e-16
   8.28966525e-17  4.73695157e-17  4.73695157e-17  7.10542736e-17
   0.00000000e+00 -1.18423789e-17]
 [ 8.28966525e-17  4.73695157e-17  0.00000000e+00  1.18423789e-17
  -3.55271368e-17  0.00000000e+00  2.36847579e-17 -7.10542736e-17
   4.73695157e-17  1.42108547e-16]]
```

```
matrix = sio.loadmat('sfm_points.mat')['image_points']
# print('Image Points matrix is:')
# print(matrix.shape)
tv = np.mean(matrix,axis=1)
print('Translation Vector t is:\n',tv)
```

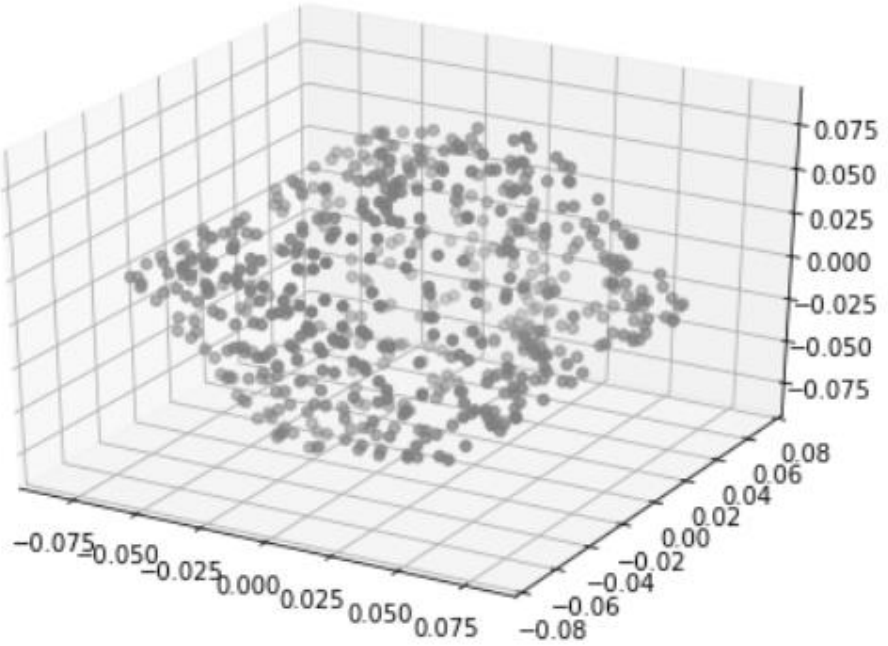
```
Translation Vector t is:
 [[ 5.49560397e-17  3.31216536e-17 -1.06118817e-16  4.27435864e-17
  -9.25185854e-19 -7.75305746e-17  1.22124533e-17  4.99600361e-18
  -9.43689571e-18 -2.08166817e-18]
 [-7.03141249e-18  3.97829917e-18  1.24900090e-17 -1.85962357e-17
  -3.99217696e-17  8.83552490e-17  6.36527867e-17  2.71773345e-18
  -2.14643118e-17  3.85802501e-17]]
```

```
#for center matrix
cm = np.zeros(matrix.shape)
for i in range(cm.shape[0]):
    for j in range(cm.shape[2]):
        cm[i,:,j] = matrix[i,:,j] - tv[i][j]
#for measure matrix
mm = np.zeros([matrix.shape[0]*matrix.shape[2],matrix.shape[1]])
for i in range(matrix.shape[1]):
    for j in range(matrix.shape[2]):
            mm[2*j,i] = cm[0,i,j]
            mm[2*j+1,i] = cm[1,i,j]

Ut, Dt, Vt = np.linalg.svd(mm)
Vt = Vt.T
Dt = Dt * np.identity(Dt.shape[0])
# print (Ut.shape)
# print (Dt.shape)
# print (Vt.shape)
location = np.dot(Ut[:,:3],Dt[:3,:3])
print('location for cameras:\n',location)
Mi = np.matmul(Ut[:, :3], np.diag(Dt[:3]))
print("Mi_shape :", Mi.shape)
```

```
location for cameras:
 [[-7.50914219  3.30837904 -3.71763726]
 [-4.53754376 -1.57773527  7.74574759]
 [ 0.17858821 -8.56620251 -2.47587867]
 [ 9.05169424  0.12603637  0.70587237]
 [ 8.25306132  2.16911022 -3.48212517]
 [-0.13132314 -7.68175234 -4.32518806]
 [-3.76826539 -8.34775199  1.20087007]
 [ 8.27600638 -3.50666717  0.57004455]
 [-0.73461089 -8.39784553 -2.88977146]
 [-8.50036578  1.60529571 -2.55252038]
 [ 8.45690903 -2.56525708 -1.79392742]
 [-3.28948312 -6.10374195 -5.44642826]
 [-2.96665571 -7.78843781 -3.22986642]
 [ 8.45107965 -1.64131526 -2.78078037]
 [-1.4368307  -8.62307292  3.07678742]
 [-7.95142326 -0.23710514 -4.1742912 ]
 [ 8.6277954  -2.12325785 -1.6361374 ]
 [-0.41749971  4.10544054 -8.14813897]
 [ 7.44257036 -3.77728996  3.4002285 ]
 [-5.22854825 -5.82482627  5.11580038]]
Mi_shape : (20,)
```

```
points = Vt[:,:3]
pylab.ion()#https://www.programcreek.com/python/example/60410/pylab.ion
Axes3D(pylab.figure()).scatter3D(points[:,0],points[:,1],points[:,2],marker='o',color='grey')
```

<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7fb634250850>




```
print('First 10 cameras locations are: \n',location[:10])
print('First 10 translation vectors \n',tv[:10])
print('3d coordinates of first 10 world points\n',points[:10])
```

```
First 10 cameras locations are:
 [[-7.50914219  3.30837904 -3.71763726]
 [-4.53754376 -1.57773527  7.74574759]
 [ 0.17858821 -8.56620251 -2.47587867]
 [ 9.05169424  0.12603637  0.70587237]
 [ 8.25306132  2.16911022 -3.48212517]
 [-0.13132314 -7.68175234 -4.32518806]
 [-3.76826539 -8.34775199  1.20087007]
 [ 8.27600638 -3.50666717  0.57004455]
 [-0.73461089 -8.39784553 -2.88977146]
 [-8.50036578  1.60529571 -2.55252038]]
First 10 translation vectors
 [[ 5.49560397e-17  3.31216536e-17 -1.06118817e-16  4.27435864e-17
  -9.25185854e-19 -7.75305746e-17  1.22124533e-17  4.99600361e-18
  -9.43689571e-18 -2.08166817e-18]
 [-7.03141249e-18  3.97829917e-18  1.24900090e-17 -1.85962357e-17
  -3.99217696e-17  8.83552490e-17  6.36527867e-17  2.71773345e-18
  -2.14643118e-17  3.85802501e-17]]
3d coordinates of first 10 world points
 [[ 0.00577163  0.06460628 -0.02497615]
 [ 0.0005761   0.06885363 -0.03458151]
 [-0.04293585  0.06330479  0.02861711]
 [ 0.04745038  0.04904207 -0.01257547]
 [-0.04210186  0.06789239  0.01175164]
 [ 0.05961964  0.0460518  -0.01438374]
 [ 0.00909167  0.06002049 -0.01229997]
 [ 0.01039489  0.04602065  0.03529275]
 [-0.02589081  0.05702972  0.03337375]
 [ 0.01745598  0.04054264  0.04731859]]
```