

Tianyu Zhang tz1280@nyu.edu (mailto:tz1280@nyu.edu) Assignment 3

```
In [1]: 1 import numpy as np
        2 from matplotlib import pyplot as plt
        3 import random
        4 import cv2
        5 from scipy.linalg import null_space
        6 import scipy.io as sio
        7 from mpl_toolkits.mplot3d import *
        8 import matplotlib.pyplot as plt
```

executed in 352ms, finished 00:48:44 2019-12-22

Problem 1

First run the SIFT detector over both images to produce a set of regions, characterized by a 128d descriptor vector. Display these regions on each picture to ensure that a satisfactory number of them have been extracted. Please include the images in your report.

```
In [2]: 1 images = ['book.pgm', 'scene.pgm']
        2 def SIFT(image):
        3     img = cv2.imread(image)
        4     img_raw = img
        5     sift = cv2.xfeatures2d.SIFT_create()
        6     kps, descs = sift.detectAndCompute(img, None)
        7     img_disp=cv2.drawKeypoints(img,kps, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
        8     return img_raw, img_disp, kps, descs
        9
        10 img_raw1, img_disp1, kp1, des1 = SIFT(images[1])
        11 img_raw2, img_disp2, kp2, des2 = SIFT(images[0])
        12 info1 = img_disp1, kp1, des1
        13 info2 = img_disp2, kp2, des2
```

executed in 72ms, finished 00:48:44 2019-12-22

When we are not using Jupyter, we can plot the image by cv2 package (see below), but using matplotlib is more convenient in Jupyter Notebook

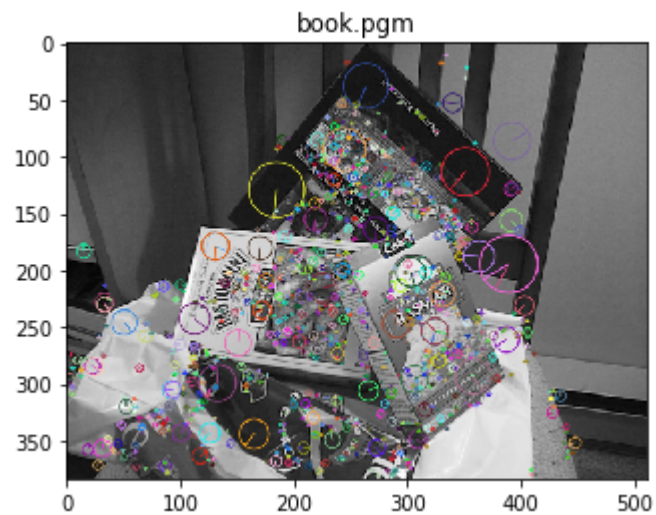
```
cv2.startWindowThread()  
cv2.imshow('image',img1)  
cv2.waitKey(5000)  
# waiting for 5 seconds for screenshot  
cv2.destroyAllWindows()
```

```
cv2.startWindowThread()  
cv2.imshow('image',img2)  
cv2.waitKey(5000)  
# waiting for 5 seconds for screenshot  
cv2.destroyAllWindows()
```

In [3]:

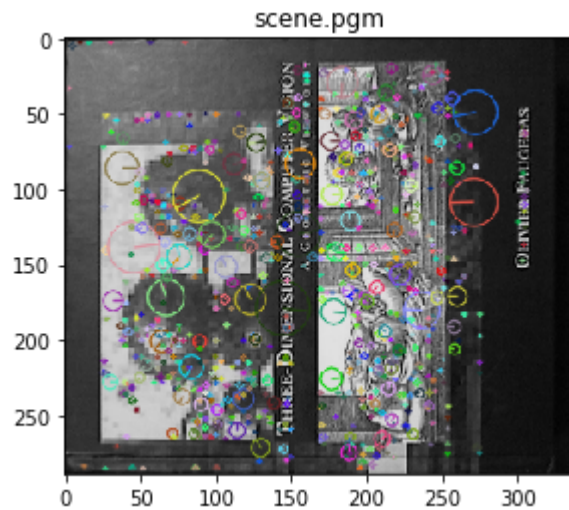
```
1 plt.imshow(img_disp1)  
2 plt.title('book.pgm')  
3 plt.show()
```

executed in 169ms, finished 00:48:44 2019-12-22



```
In [4]: 1 plt.imshow(img_disp2)
        2 plt.title('scene.pgm')
        3 plt.show()
```

executed in 141ms, finished 00:48:44 2019-12-22



Obtain a set of putative matches T and test the functioning of RANSAC, plot out the two images side-by-side with lines showing the potential matches

```
In [5]: 1 def Matcher(info1, info2, threshold = 0.9, Nfilter_ = -1, flags = 0):
2
3     img1, kp1, des1 = info1
4     img2, kp2, des2 = info2
5
6     bf = cv2.BFMatcher(cv2.NORM_L2)
7     matches = bf.knnMatch(queryDescriptors=des1,trainDescriptors=des2, k=2)
8     good = []
9     for m,n in matches:
10         if m.distance < threshold*n.distance:
11             good.append(m)
12     filter_ = sorted(good, key = lambda x: x.distance)
13     img3 = cv2.drawMatches(img1,kp1,img2,kp2,filter_[0:Nfilter_],None,flags=flags)
14     print("# matches shown in the following picture: ",len(filter_[0:Nfilter_]))
15     plt.figure(figsize=(15,20))
16     plt.imshow(img3)
17     plt.show()
18     return good, filter_[0:Nfilter_], img3
```

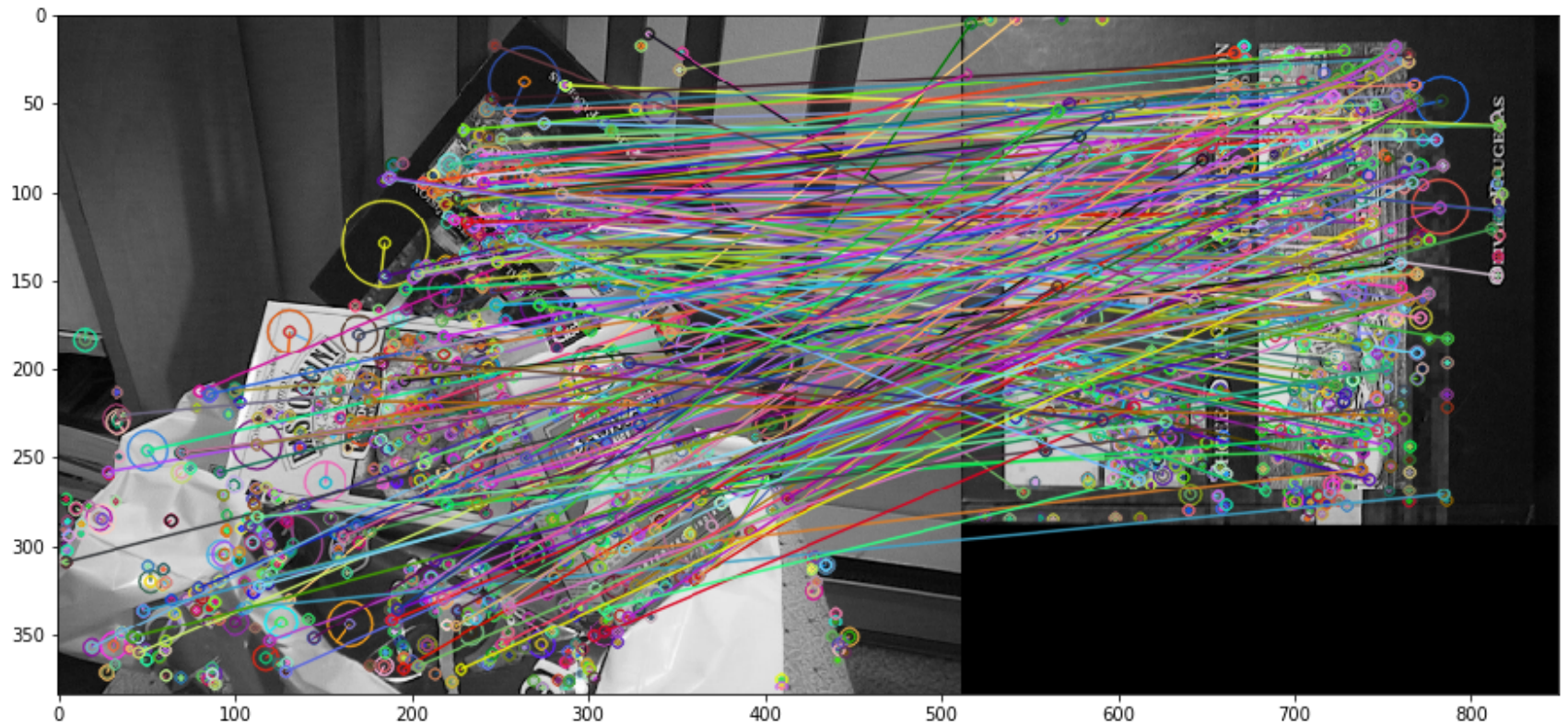
executed in 7ms, finished 00:48:44 2019-12-22

This is the result for all the matches

```
In [6]: 1 good, _, _ = Matcher(info1, info2, threshold = 0.9, Nfilter_ = None)
```

executed in 369ms, finished 00:48:44 2019-12-22

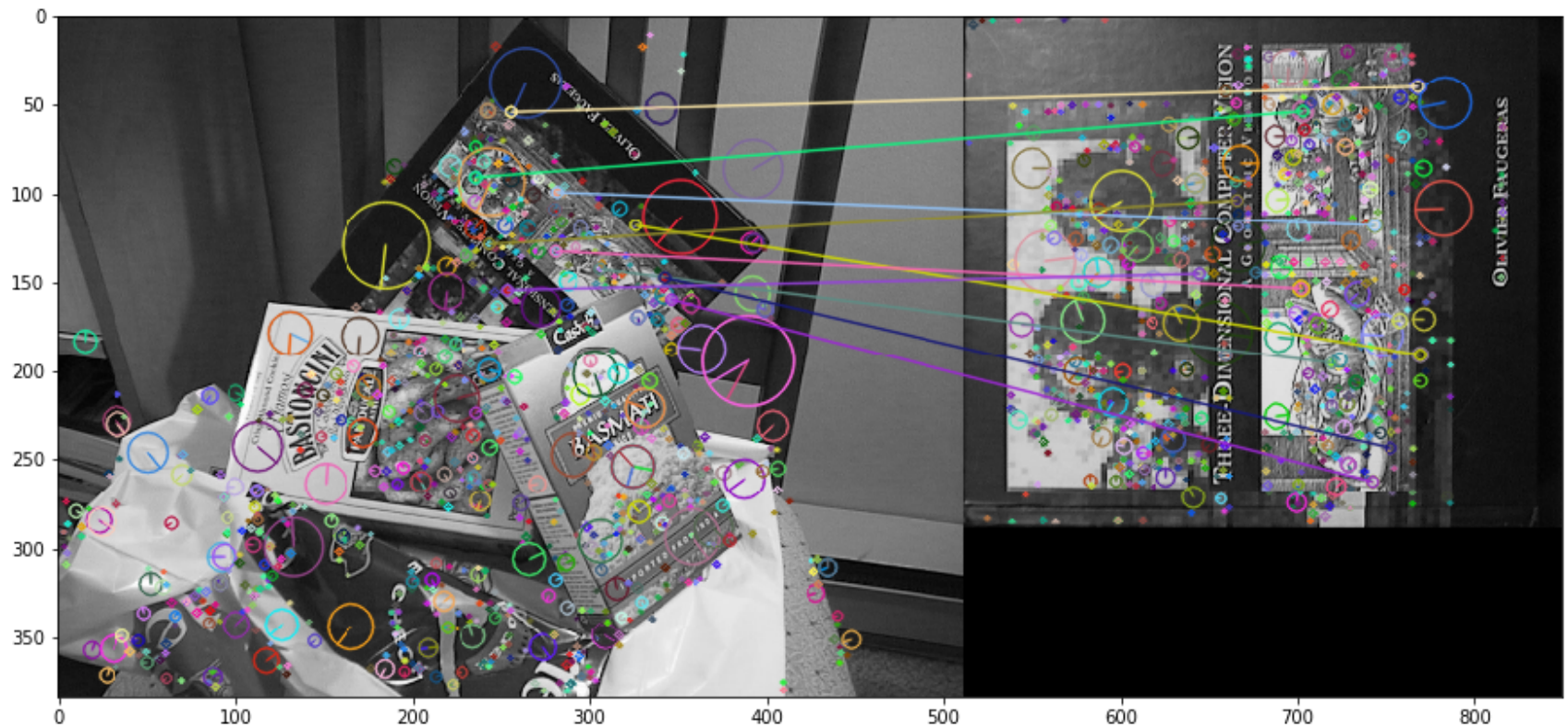
matches shown in the following picture: 248




```
In [7]: 1 good, _, _ = Matcher(info1, info2, threshold = 0.9, Nfilter_ = 10, flags=2)
```

executed in 362ms, finished 00:48:45 2019-12-22

matches shown in the following picture: 10



RANSAC Part to improve the result from the SIFT:

Repeat N times (where N is ~100):

- Pick P matches at random from the total set of matches T. Since we are solving for an affine transformation which has 6 degrees of freedom, we only need to select P=3 matches.

- Construct a matrix A and vector b using the 3 pairs of points as described in lecture 6.
- Solve for the unknown transformation parameters q .
- Using the transformation parameters, transform the locations of all T points in image 1. If the transformation is correct, they should lie close to their pairs in image 2.
- Count the number of inliers, inliers being defined as the number of transformed points from image 1 that lie within a radius of 10 pixels of their pair in image 2.
- If this count exceeds the best total so far, save the transformation parameters and the set of inliers.
- End repeat.

The refit process:

- Perform a final refit using the set of inliers belonging to the best transformation. This refit should use all inliers, not just 3 points.

In [8]:

```
1 good = np.array(good)
```

executed in 4ms, finished 00:48:45 2019-12-22

In [9]:

```
1 def aug(arr):
2     '''
3     This function is for augment the array that will be used in this problem
4     '''
5     arr0 = arr
6     aug = np.ones((arr.shape[0],arr0.shape[1],1))
7     return np.append(arr0, aug, axis=2)
8
9 def getAugQorT(good, kp1, kp2, isQuery = True):
10    """
11    This function is to get the augmented version of descriptors and descriptors
12    """
13    if isQuery:
14        tmp = np.array([kp1[m.queryIdx].pt for m in good])
15    else:
16        tmp = np.array([kp2[m.trainIdx].pt for m in good])
17    return aug(tmp.reshape(-1,1,2))
18 def getRANSACsource(good, kp1, kp2):
19    """
20    This function is to integrate the data that the RANSAC needed
21    """
22    augSrcPts = getAugQorT(good, kp1, kp2, True)
23    augDstPts = getAugQorT(good, kp2, kp1, False)
24    Rsource = []
25    for i in range(len(augSrcPts)):
26        Rsource.append([augSrcPts[i], augDstPts[i]])
27    return np.array(Rsource)
```

executed in 10ms, finished 00:48:45 2019-12-22

In [10]:

```
1 Rsource = getRANSACsource(good, kp1, kp2)
2 Rsource = list(Rsource)
3 def runRANSAC(img1, img2, data):
4     """
5     This function runs the RANSAC and the refit process
6     """
7     # phase one
8     bestcount = 0
9     for i in range(100): # repeat N times
10         inL = [] # inliers
11         # pick P matches without replacement
12         sample = random.sample(data, 3)
13         coefMat = np.zeros((1,6))
14         tarVec = np.zeros((1,1))
15         # build up the linear system to solve the unknown transformation parameters q
16         for i in range(3):
17             temp = np.append(sample[i][0], np.zeros((1,3)), axis=1)
18             temp1 = np.append(np.zeros((1,3)), sample[i][0], axis=1)
19             coefMat = np.append(coefMat,temp, axis=0)
20             coefMat = np.append(coefMat,temp1, axis=0)
21             tarVec = np.append(tarVec, sample[i][1].reshape((3,1))[:-1], axis =0)
22         coefMat = coefMat[1:, :]
23         tarVec = tarVec[1:, :]
24         try:
25             # the coefficient matrix is not gauranteed to be full rank, thus, the solution will not always
26             sol = np.linalg.solve(coefMat, tarVec)
27         except:
28             sol = None
29             continue
30         count = 0
31         # judge the inliers
32         for j in range(len(data)):
33             a = sol.reshape((2,3)).dot(data[j][0].T).reshape((1,2))
34             b = np.delete(data[j][1], 2, axis=1).reshape((1,2))
35             if np.linalg.norm(a-b) < 10:
36                 count += 1
37                 inL.append(data[j])
38         # get the best models and inliers
39         if count > bestcount:
40             bestcount = count
41             bestmodel = sol
42             bestinliners = inL
```

```

43 bestmodel = bestmodel.reshape((2,3))
44 # phase 2
45 refit_model = np.zeros((6,1))
46 count_singularity = 0
47 for i in range(len(bestinliners)-2):
48     refit_list = [bestinliners[i], bestinliners[i+1], bestinliners[i+2]]
49     # get the refit model by the refit list by solving linear system
50     coefMat = np.zeros((1,6)) # coefficient Matrix
51     tarVec = np.zeros((1,1)) # target Matrix
52     for i in range(3):
53         temp = np.append(refit_list[i][0], np.zeros((1,3)), axis=1)
54         temp1 = np.append(np.zeros((1,3)), refit_list[i][0], axis=1)
55         coefMat = np.append(coefMat,temp, axis=0)
56         coefMat = np.append(coefMat,temp1, axis=0)
57         tarVec = np.append(tarVec, refit_list[i][1].reshape((3,1))[:-1], axis =0)
58     coefMat = coefMat[1:, :]
59     tarVec = tarVec[1:, :]
60     try:
61         # the coefficient matrix is not gauranteed to be full rank, thus, the solution will not always
62         sol = np.linalg.solve(coefMat, tarVec)
63     except:
64         sol = None
65         continue
66     sol = np.linalg.solve(coefMat, tarVec)
67     refit_model += sol
68 refit_model = refit_model / (len(bestinliners) - count_singularity)
69 refit_model = refit_model.reshape((2, 3))
70 return bestcount,refit_model,bestmodel

```

executed in 27ms, finished 00:48:45 2019-12-22

```

In [11]: 1 result = runRANSAC(img_raw1, img_raw2, Rsource)
         2 print('Best number of inliers:', result[0])

```

executed in 303ms, finished 00:48:45 2019-12-22

Best number of inliers: 119

Homography matrix H

```
In [12]: 1 print("H_Matrix:",result[1])
```

executed in 5ms, finished 00:48:45 2019-12-22

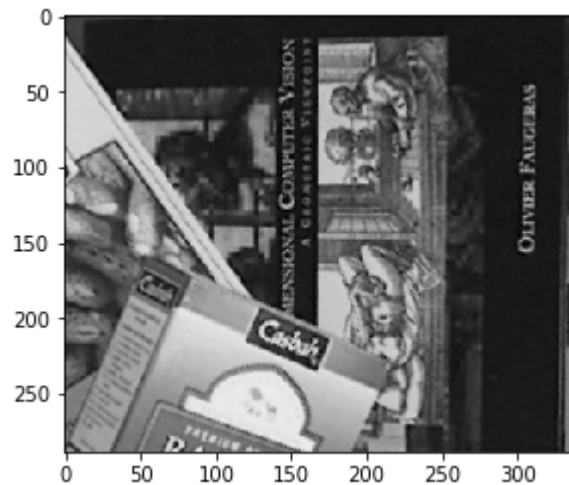
```
H_Matrix: [[ 0.7426577 -0.83094475 49.45955145]
 [ -1.52112024  0.92086412 471.61425534]]
```

Image transformation

- Finally, transform image 1 using this final set of transformation parameters, q. Use the cv2.warpAffine from the OpenCV-Python environment. Display this image and find that the pose of the book in the scene should correspond to its pose in image 2.

```
In [13]: 1 pose = cv2.warpAffine(img_raw1, result[2], (img_raw2.shape[1], img_raw2.shape[0]))
2 plt.imshow(pose)
3 plt.show()
```

executed in 143ms, finished 00:48:45 2019-12-22



Problem 2

Loading the data

```
In [14]: 1 world = np.loadtxt("world.txt")
2 image = np.loadtxt("image.txt")
3 len_ = len(world)
4
5 def agument(input_arr):
6     input_arr = np.array(input_arr)
7     Shape_ = input_arr.shape
8     aug_arr = np.ones((1,Shape_[-1]))
9     return np.append(input_arr, aug_arr, axis=0)
10
11 aguWorld = agument(world)
12 aguImg = agument(image)
13
14 zero_Vec = np.zeros((4, 1))
15 A = np.zeros((1,12))
```

executed in 7ms, finished 00:48:45 2019-12-22

Calculating the P matrix by a series of linear equations

$$\begin{bmatrix} O^T & -w_i \mathbf{X}_i^T & y_i \mathbf{X}_i^T \\ w_i \mathbf{X}_i^T & O^T & -x_i \mathbf{X}_i^T \\ -y_i \mathbf{X}_i^T & x_i \mathbf{X}_i^T & O^T \end{bmatrix} \begin{pmatrix} P^1 \\ P^2 \\ P^3 \end{pmatrix} = 0$$

for each correspondence $\mathbf{x}_i \leftrightarrow \mathbf{X}_i$, where $\mathbf{x}_i = (x_i, y_i, w_i)^T$, w_i being the homogeneous coordinate, and P^j is the j^{th} row of P . But since the 3rd row is a linear combination of the first two, we need only consider the first two rows for each correspondence i . Thus, you should form a 20 by 12 matrix A , each of the 10 correspondences contributing two rows. This yields $Ap = 0$, p being the vector containing the entries of matrix P .

To solve for p , we need to impose an extra constraint to avoid the trivial solution $p = 0$. One simple one is to use $\|p\|_2 = 1$. This constraint is implicitly imposed when we compute the SVD of A . The value of p that minimizes Ap subject to $\|p\|_2 = 1$ is given by the eigenvector corresponding to the smallest singular value of A . To find this, compute the SVD of A , picking this eigenvector and reshaping it into a 3 by 4 matrix P .

In [15]:

```
1  for i in range(10):
2      xi_ = aguImg[:, i].reshape(3,1)
3      Xi = aguWorld[:, i].reshape(4,1)
4      part1 = np.concatenate((zero_Vec.T, -1*xi_[2] * Xi.T, xi_[1] * Xi.T), axis=1)
5      part2 = np.concatenate((-1*xi_[2] * Xi.T, zero_Vec.T, xi_[0] * Xi.T), axis=1)
6      A = np.append(A, part1, axis=0)
7      A = np.append(A, part2, axis=0)
8
9  # remove the first zeros
10 A = A[1:,:]
11
12 # Estimation of the P matrix
13 P = np.linalg.svd(A)[2][:-1, :].reshape((3, 4))
14 print ("Estimation of Camera Matrix P:")
15 print (P)
```

executed in 7ms, finished 00:48:45 2019-12-22

Estimation of Camera Matrix P:

```
[[ 1.27000127e-01  2.54000254e-01  3.81000381e-01  5.08000508e-01]
 [ 5.08000508e-01  3.81000381e-01  2.54000254e-01  1.27000127e-01]
 [ 1.27000127e-01 -4.16333634e-17  1.27000127e-01 -5.55111512e-17]]
```

Two ways to estimation the center C:

Method 1:

Now we have P, we can compute the world coordinates of the projection center of the camera C. Note that $PC = 0$, thus C lies in the null space of P, which can again be found with an SVD (the Matlab command is `svd`). Compute the SVD of P and pick the vector corresponding to this null-space. Finally, convert it back to inhomogeneous coordinates and to yield the (X,Y,Z) coordinates. Your report should contain the matrix P and the value of C.

```
In [16]: 1 # Method1 : Estimation of the Center
2 C = np.linalg.svd(P)[2][:-1, :]
3 C = (C/C[-1])[:-1]
4 print ("Estimation of Projection Center C:")
5 print (C)
```

executed in 5ms, finished 00:48:45 2019-12-22

Estimation of Projection Center C:
[1. -1. -1.]

In the alternative route, we decompose P into its constituent matrices. Recall from the lectures that $P = K[R|t]$. However, also, $t = -R\hat{C}$, \hat{C} being the inhomogeneous form of C . Since K is upper triangular, use a RQdecomposition to factor KR into the intrinsic parameters K and a rotation matrix R . Then solve for \hat{C} . Check that your answer agrees with the solution from the first method.

```
In [17]: 1 # Method2 : Estimation of the Center
2 R = -1*np.linalg.qr(P.T)[0][:-1,:].T
3 RChat = np.linalg.qr(P.T)[0][-1,:].T
4 C = np.linalg.solve(R, RChat)
5 print("C_hat estimation (QR Decomposition):", C)
```

executed in 6ms, finished 00:48:45 2019-12-22

C_hat estimation (QR Decomposition): [1. -1. -1.]

Problem 3

Loading the data

```
In [18]: 1 sfm = sio.loadmat('sfm_points.mat')
2 center = np.zeros((2,1))
3 W = np.zeros((20, 600))
```

executed in 5ms, finished 00:48:45 2019-12-22

We do this in the following stages:

- Compute the translations t_i directly by computing the centroid of point in each image i .

- Center the points in each image by subtracting off the centroid, so that the points have zero mean
- Construct the $2m$ by n measurement matrix W from the centered data.

```
In [19]: 1 for i in range(10):
2         tmp = sfm["image_points"][ :, :, i]
3         len_ = tmp.shape[1]
4         x = np.sum(tmp[0, :])/len_
5         y = np.sum(tmp[1, :])/len_
6         points = np.array([x,y])
7         center = np.append(center, points.reshape(-1,1), axis=1)
8         # exclude the first zero
9         #print(center)
10        center = center[:, 1:]
11        #print(center)
12
13        for i in range(10):
14            part1 = sfm["image_points"][ :, :, i]
15            part2 = center[:, i].reshape(2,1)
16            tmp = part1 - part2
17            W[i] = tmp[0, :]
18            W[i+10] = tmp[1, :]
19        #print(W)
20        print ("t_i (first camera):", center[:,0])
```

executed in 10ms, finished 00:48:45 2019-12-22

t_i (first camera): [2.36847579e-17 8.28966525e-17]

- Perform an SVD decomposition of W into UDV^T .
- The camera locations M^i can be obtained from the first three columns of U multiplied by $D(1 : 3, 1 : 3)$, the first three singular values.

```
In [20]: 1 from numpy.linalg import cond
2         cond(W)
```

executed in 8ms, finished 00:48:45 2019-12-22

Out[20]: 4.837719320731832e+16

```
In [21]: 1 M_i = np.dot(np.linalg.svd(W)[0][:, :3], np.diag(np.linalg.svd(W)[1][:3]))
          2 print ("M_i (first camera):", M_i[:2,:])
```

executed in 29ms, finished 00:48:45 2019-12-22

```
M_i (first camera): [[-7.50914219  3.30837904 -3.71763726]
 [ 0.17858821 -8.56620251 -2.47587867]]
```

```
In [22]: 1 M_i[:,2,:]
```

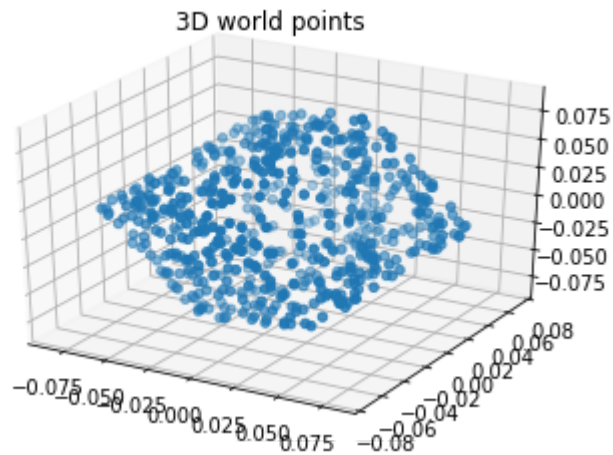
executed in 4ms, finished 00:48:45 2019-12-22

```
Out[22]: array([[ -7.50914219,  3.30837904, -3.71763726],
 [  0.17858821, -8.56620251, -2.47587867]])
```

- The 3D world point locations are the first three columns of V .
- Verify the answer by plotting the 3D world points out. using the plot3 command. The rotate3d command will let you rotate the plot. This functionality is replicated in Python within the matplotlib package.

```
In [23]: 1 fig = plt.figure()
2 ax = plt.axes(projection='3d')
3 ax.set_title('3D world points ')
4 # Perform an SVD decomposition of W
5 px,py,pz = np.linalg.svd(W)[2][:3]
6 ax.scatter(px, py, pz)
7 plt.show()
```

executed in 148ms, finished 00:48:46 2019-12-22



```
In [24]: 1 print ("3D coordinates of the first 10 world points")
2 print (np.matrix.transpose(np.linalg.svd(W)[2][:3][:, 0:10]))
```

executed in 15ms, finished 00:48:46 2019-12-22

```
3D coordinates of the 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]]
```

