ECE 661 - Homework-7

Vishveswaran Jothi
vjothi@purdue.edu

10-30-2016

1 Theory and Implementation

1.1 Introduction

The given task is to perform Image Registration on two depth images. This assignment introduces many plotting techniques along with technique that are used to create point clouds. Creating point clouds are vital in 3D mapping. As many research activities in computer vision are in 3D. This assignment will be a base for many 3D computation. The input for this assignment is obtained from kinect 2 sensor.

1.2 Creating Point Cloud

The following algorithm gives a brief overview to generate a point cloud from a depth image.
Step1: Get the path of the depth images and their names from the user. Also get the "K" and threshold from the user.
Step2: Load the text file of the depth image in a python script. The values in the depth image file is the distance from the sensor not the RGB values.
Step3: Save the images as color depth map using imshow and savefig modules in matplotlib package in python.
Step4: Create a point cloud for the given image map by using the below formula.

$$Pointcloud(u) = Depth(u) * K^{-1} * u$$

where u is in homogenous coordinates $(x, y, 1)^T$, D(u) is the depth value of that coordinate and K is given by intrinsic parameter of the camera.

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

Note that in the given K matrix s=0.
Step5: The above obtained point cloud is a 3D point. The computed point cloud is plotted using the algorithm explained in the plotting section.

1.3 Iterative Closest Point (ICP) - Algorithm

In this section, ICP is explained algorithmic manner, since theory is extensively covered in class. But necessary mathematical equation will be provided to support the algorithm.
Step1: Find the point cloud of the given pair of image using above algorithm.
Step2: Now we need to find the correspondence of the two images.
Step3: To find the correspondence, First find the euclidean distance between all the 3D points in both the images.
Euclidean= $\sqrt[2]{(x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2}$ **Step4:** Choose the value in image 2 which has smallest euclidean distance w.r.to image 1.
Step5: Also we should remove the many to one correspondence in the mapping function by checking the index of the minimum euclidean with the list of previous indexes of image 2 which has a domain in image1.
Step6: After the correspondence, Find the Centroid of the correspondence by the given formula.

$$Image_1 centriod = \Sigma_1^N (P_i')/N$$

where N is the total no.of correspondence, P_i'= each 3D point in the correspondence.
Step7: Similarly, Find the Centroid of Image2. **Step8:** Now compute the difference matrix for both images using the given formula.

$$M_I mage1 = P_i' - {P_{centroid}}_1^N$$

Repeat the same for Image 2 as well.
Step9: Compute the correlation matrix (C) from difference matrix of both images, say M_p, M_q.

$$C = M_q * M_p^T$$

If M_q and M_q are 3XN, but in my algorithm it is given as $M_q^T * M_p$, since they are considered as NX3.
Step10: The Rotation matrix that relates the Image1 and image2 is given by the SVD of C.

$$R = V * U^T$$

where U is the orthogonal matrix formed by eigen vectors of AA^T and the V is the orthonormal matrix for med by eigen vectors of $A^T A$
Step11: The Translation vector that relates the Image1 and Image2 is obtained from the difference of the centroid of Image1 with the product of Rotation matrix with centroid of Image2.

$$trans_v ec(T) = P_c' entroid - R * Q_c' entroid$$

Step12: The Homogenous transformation matrix is given by ...

$$H = \begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix}$$

Step13: Now set transform the Image 2 point clouds with the H matrix.
Step14: Repeat the above procedure from Step 2 to Step 14 for the user specified iterations. But set the point cloud of Image 2 as the transformed point cloud values obtained as the result of each iteration.
Step15: Finally plot the resultant point clouds using the MatplotLib package using the algorithm in the next section.

1.4 Plotting using MatplotLib

A brief explanation to use the matplotlib package which might act as a simple guidance for plotting 3D plots.
Step1: Get the parameter such as color for the marker from the user. Import matplotlib and mpl_toolkits.mplot3d to plot in 3D space.
Step2: Create a figure with a figure size (say) 20X30 smaller or standard size will not be sufficient to analyse the output.
Step3: Now create a 3D plot with axes set as '3D'
Step4: Pass the x,y,z axes values along with edgecolor, markersize, marker color in the scatter plot module.
Step5: Repeat the same for all the points in both image.
Step6: Now to rotate the image so that to make it easy to visualize, use view_init module for that. Note: Look for the comment part in the plotting code for sample demonstration.
Step7: Finally save it using savefig module.

1.5 Performance of ICP

The performance of ICP is compared with the result obtained from plotting the point cloud of both images obtained from the point cloud algorithm.

1. Generally, The result with ICP will be better than without ICP. But in the given image we cannot say that, since both images are already significantly close by.

2. Computation of ICP in python took more time say around 7 mins and plotting took around 8 mins for each pair of image (with and without ICP).

3. After twenty iterations of ICP most of the image is merged within each other, only few parts which are not exactly matching could be seen apart.

4. To check the above phenomena, threshold with 0.01 is set and the result is obtained. Which brought the two images closer, but not as exact as with the threshold 0.1, since this threshold requires more iteration to converge.

5. Also to find the convergence, the no.of corresponding is printed after getting the correspondence output from its function. It increases in a linear manner, if given enough iteration, it will definitely converge.

6. Using Trial and Error method best iteration count was obtained as 19 for threshold 0.1 mts. For a threshold of 0.01 mts, it is 24 iterations after that the no. of. matches decreases.

7. Potential drawbacks are computation power and computation time.

2 Output Images

2.1 For Given Leafless Tree Image:

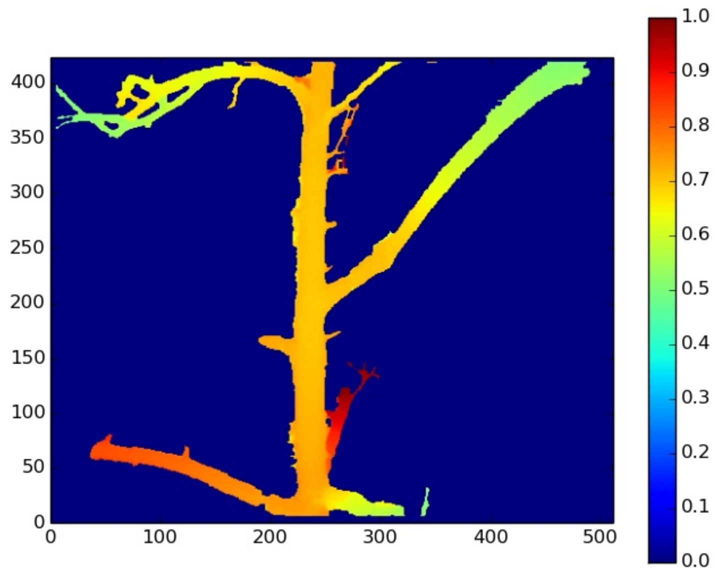

Figure 1: Input Depth Image1 of Leafless Tree

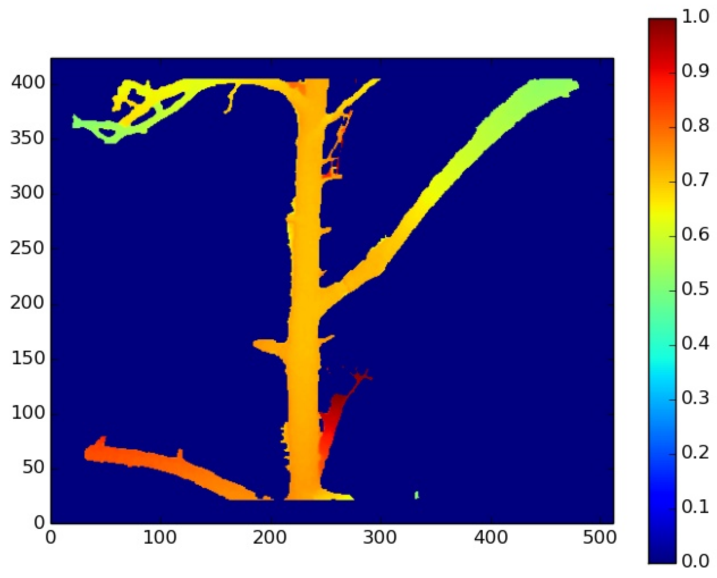

Figure 2: Input Depth Image2 of Leafless Tree

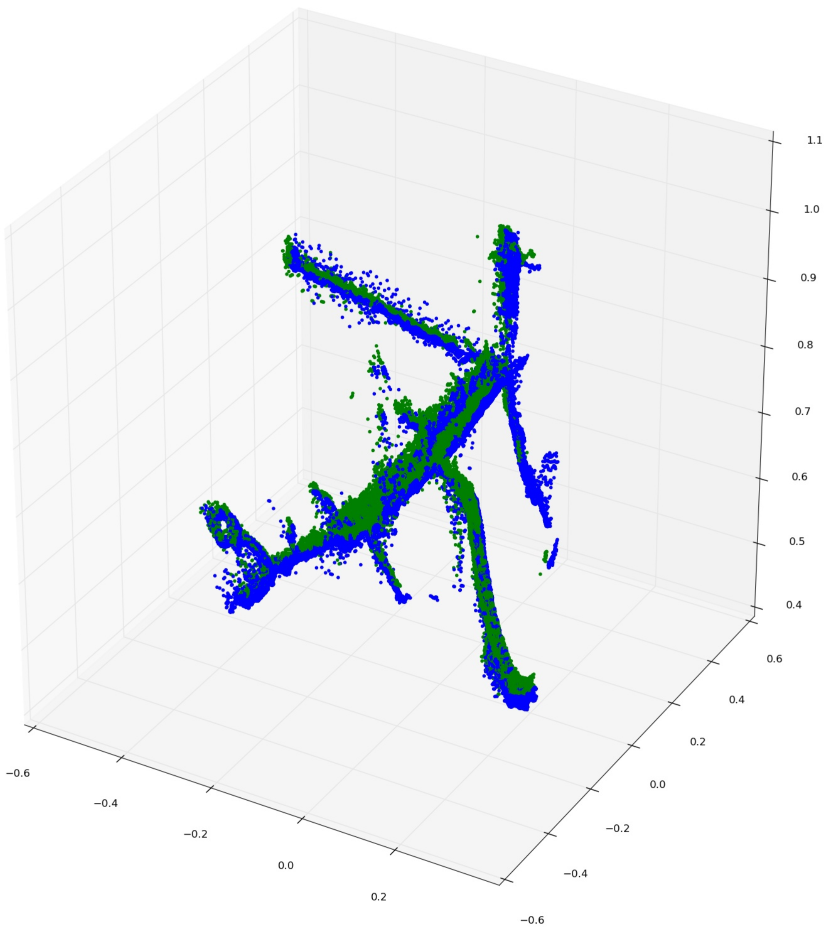

Figure 3: Point Cloud of Leafless Tree before ICP

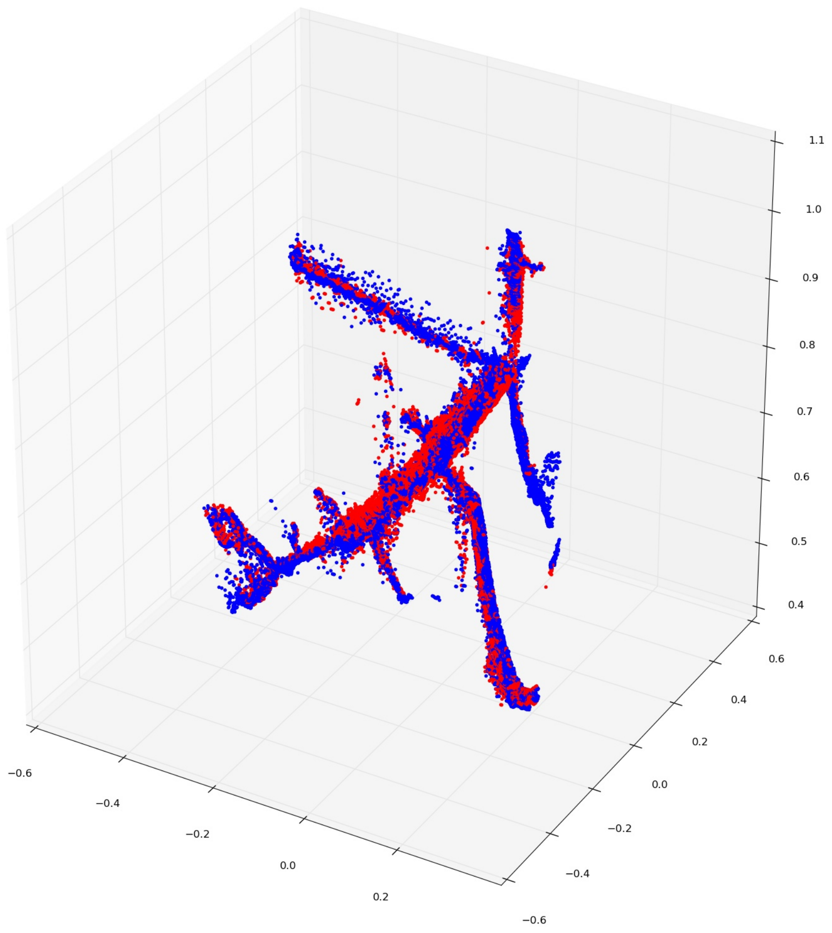

Figure 4: Point Cloud of Leafless Tree after 20 iterations of ICP

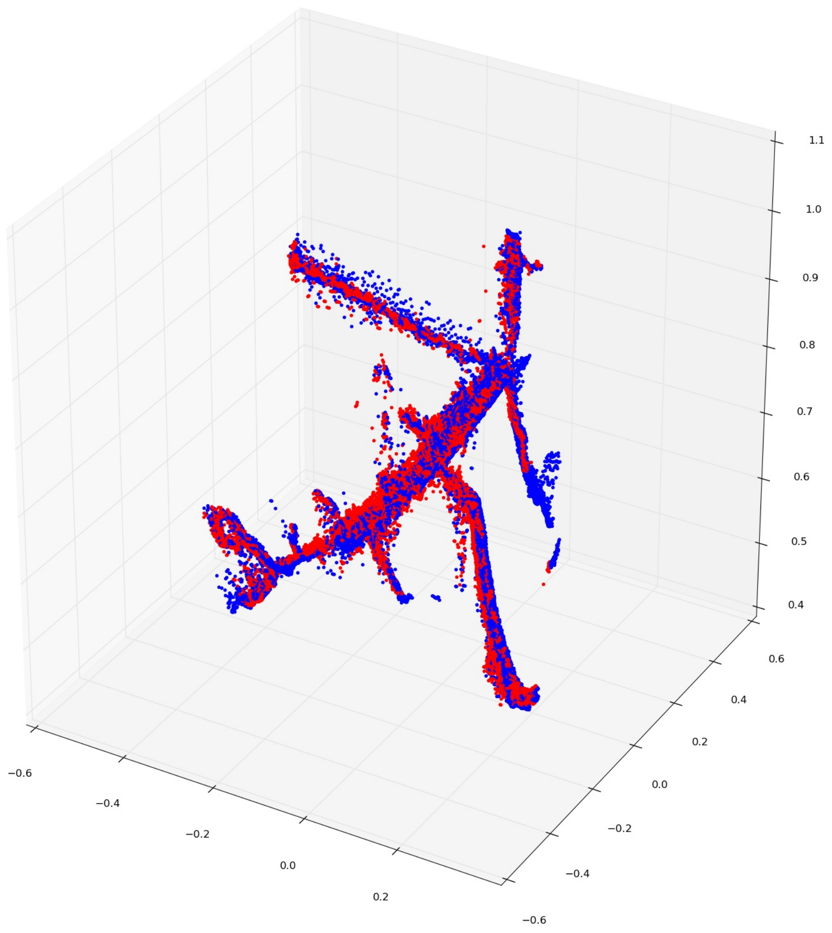

Figure 5: Point Cloud of Leafless Tree after 20 iterations of ICP including the removal many to one points

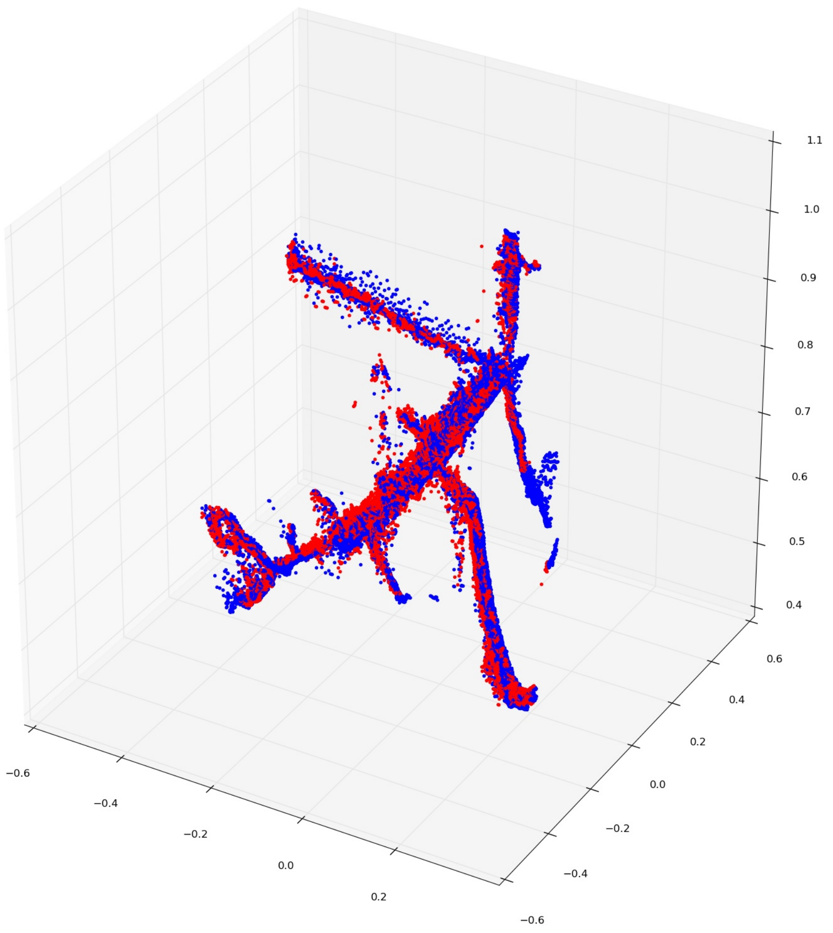

Figure 6: Point Cloud of Leafless Tree after 24 iterations of ICP including the removal many to one points

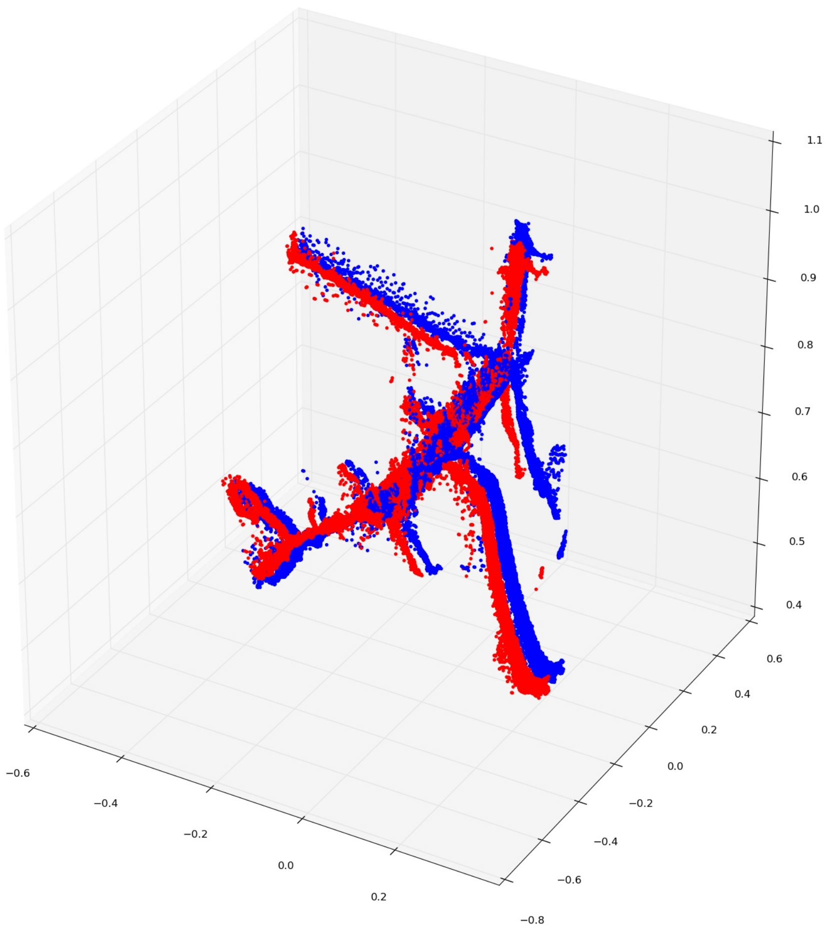

Figure 7: Point Cloud of Leafless Tree after 20 iterations of ICP with threshold as 0.01

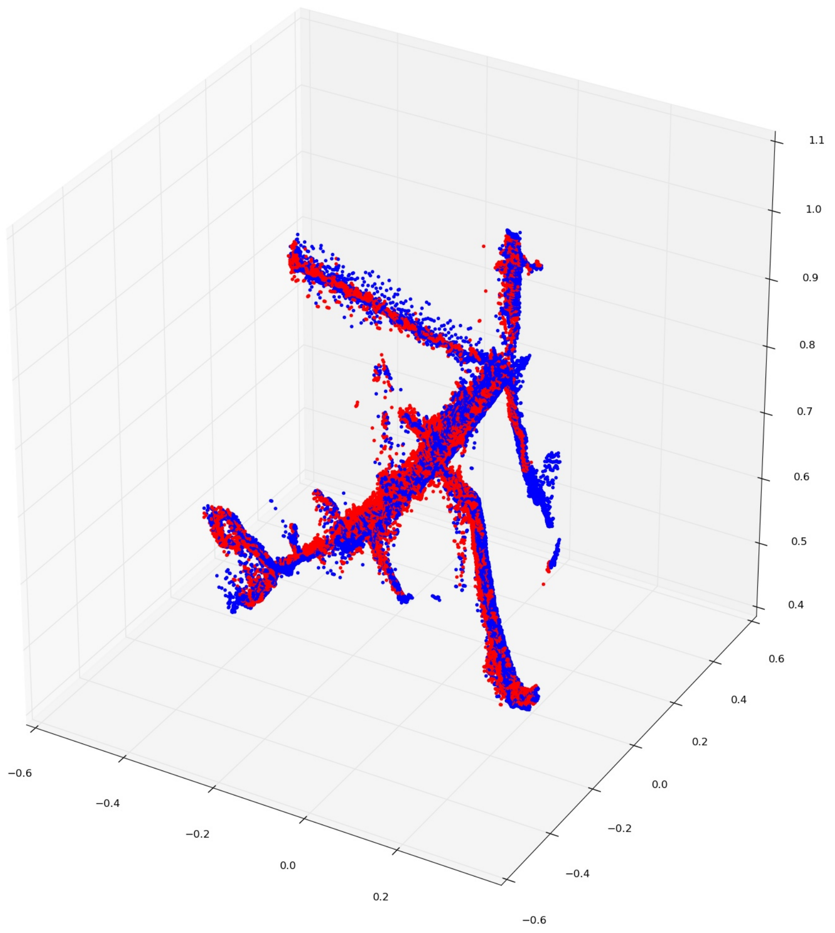

Figure 8: Point Cloud of Leafless Tree after 23 iterations of ICP with threshold as 0.01

3 Source Code

3.1 Main function

```
"""
Author: Vishveswaran Jothi
"""

import numpy as np
import matplotlib.pyplot as mpl
import pc1
import icp1
import time
#from mpl_toolkits.mplot3d import Axes3D
###########################################
# Input from the user to Load the txtfiles
###########################################

path_prompt = "Enter the full path for the text files:"
img_name_prompt = "Enter the text file name with file extension:"
max_iter_prompt = "Enter the max iteration for matching the
point clouds"
k_prompt="Enter the intrinsic parameter matrix"
threshold_prompt = "Enter the Threshold for matching correspondence"

path1=raw_input(">"+path_prompt)
img_name1=raw_input(">"+img_name_prompt)
max_iter=raw_input(">"+max_iter_prompt)
path2=raw_input(">"+path_prompt)
img_name2=raw_input(">"+img_name_prompt)
print k_prompt
K=np.zeros((3,3),dtype='int')
for loop in range(3):
    for loop1 in range(3):
        K[loop,loop1]=int(raw_input("> Enter the element"
        +[loop,loop1]+": "))
threshold=int(raw_input(">"+threshold_prompt))
"""
##########################################

##########################################
# Sample input from user
##########################################

path="/home/vishwa/661/HW7/HW7DepthIms"
img_name1="depthImage1ForHW.txt"
img_name2="depthImage2ForHW.txt"
max_iter=40
K=np.array([[365,0,256],[0,365,212],[0,0,1]])
threshold=0.1
"""
##########################################
# Algorithm starts here
##########################################
d_image1=np.loadtxt(path+"/"+img_name1)
d_image2=np.loadtxt(path+"/"+img_name2)
img_name1=img_name1.split('.')[0]
img_name2=img_name2.split('.')[0]
#cv2.imwrite('disp_'+img_name1.split('.')[0]+'.jpg',255*d_image1)
#cv2.imwrite('disp_'+img_name2.split('.')[0]+'.jpg',255*d_image2)
```

```
#create as a depth color image for Image 1
mpl.figure()
mpl.imshow(d_image1,origin='upper',extent=[0, d_image1.shape[1]
, 0, d_image1.shape[0]])
mpl.colorbar()
# Save the colored depth image
mpl.savefig('op_1'+img_name1+'.jpg')
#mpl.close()
#create as a depth color image for Image 2
mpl.figure()
mpl.imshow(d_image2,origin='upper',extent=[0, d_image2.shape[1]
, 0, d_image2.shape[0]])
mpl.colorbar()
# Save the colored depth image
mpl.savefig('op_1'+img_name2+'.jpg')
mpl.show()
#mpl.close()

##################################################
# Plot before ICP function
#################################################
# creating the point clouds
point_cloud_img1=pc1.p_cloud(d_image1,K)
point_cloud_img2=pc1.p_cloud(d_image2,K)

img_name="_before_ICP"
print "The output is saved in point_cloud"+img_name+".jpg"
pc1.display(point_cloud_img1,point_cloud_img2,'b','g',img_name)
print " Running ICP algorithm..."

##################################################
# ICP function
#################################################
#time.sleep(5)
pt_cl_2_trans=icp1.icp_alg(point_cloud_img1,point_cloud_img2,
max_iter,threshold)
print " Plotting the result of ICP algorithm..."

##################################################
# Plot after ICP function
#################################################

img_name="_after_ICP_T_0.1_var19"
print "The output is saved in point_cloud"+img_name+".jpg"
pc1.display(point_cloud_img1,pt_cl_2_trans,'b','r',img_name)
```

3.2 pc.py (Code for Finding point Cloud and Plotting)

```
"""
Author: Vishveswaran Jothi
"""

import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as mpl
#########################################
# Point cloud function starts here
#########################################
def p_cloud(d_image,K):
    # Inverting K so as P(U)=D(u)*K^-1*U
    K_inv=np.linalg.inv(K)
    point_cloud=[]
    # finding point clouds

    for loop1 in range(d_image.shape[0]):
        for loop2 in range(d_image.shape[1]):
            p_cl_tmp=d_image[loop1,loop2]*(np.dot(K_inv,
            [loop2,loop1,1]))
            if p_cl_tmp[0]!=0 or p_cl_tmp[1]!=0 or p_cl_tmp[2]!=0:
                point_cloud.append(p_cl_tmp)

    return np.asarray(point_cloud)

#########################################
# Point cloud function ends here
#########################################

#########################################
# Display the scatter plot of Images
#########################################

def display(p_cl1,p_cl2,color1,color2,img_name):
    fig = mpl.figure(figsize=(20,20))
    ax = fig.add_subplot(111, projection='3d')
    for loop in range(len(p_cl1)):
        ax.scatter(p_cl1[loop,0],p_cl1[loop,1],p_cl1[loop,2],s=9,
        c=color1,edgecolor=color1)
        #print loop

    for loop in range(len(p_cl2)):
        ax.scatter(p_cl2[loop,0],p_cl2[loop,1],p_cl2[loop,2],s=9,
        c=color2,edgecolor=color2)
    # The below commented section can be un commented to get
    rotated figure plot to compare with input
    #for angle in range(0, 360):
        #ax.view_init(270, angle)
    mpl.savefig('point_cloud'+img_name+'.jpg')
    #mpl.show()
    mpl.close()
    return
#########################################
# Display the scatter plot of Images
#########################################
```

3.3 icp.py (Code for ICP algorithm)

```
"""
Author: Vishveswaran Jothi
"""
import numpy as np
#########################################
# ICP algorithm starts here
#########################################
def icp_alg(p_cl1,p_cl2,max_iter,threshold):

    for iter in range(max_iter):
        # Printing the iteration count
        print iter
        # Finding the correspondence with euclidean distance
        p_cl1_d,p_cl2_d= Corres(p_cl1,p_cl2,threshold)


        if p_cl2_d==[]:
            print "No matches were found"
            return p_cl2_d
        # Now calculate the centroid for each image
        N=len(p_cl1_d)
        print p_cl1_d.shape,p_cl2_d.shape
        # Finding centroid
        p_centroid=np.array([sum(p_cl1_d[:,0]),sum(p_cl1_d[:,1]),
        sum(p_cl1_d[:,2])])/(N)
        q_centroid=np.array([sum(p_cl2_d[:,0]),sum(p_cl2_d[:,1]),
        sum(p_cl2_d[:,2])])/(N)
        print p_centroid,q_centroid

        # Now calculate resulting point clouds
        Mp=np.subtract(p_cl1_d,p_centroid)
        Mq=np.subtract(p_cl2_d,q_centroid)
        C=np.dot(Mq.T,Mp)
        # to find the rotation and translation between the images
        U,sig,Vt=np.linalg.svd(C)
        Rot_mat_tmp=np.dot(U,Vt)
        Rot_mat=Rot_mat_tmp.T
        # obtain the translation by P_centroid-R*q_centroid
        trans_vec=p_centroid-np.dot(Rot_mat,q_centroid)
        tfm_mat=np.zeros((4,4),dtype='float')
        tfm_mat[0:3,0:3]=Rot_mat
        tfm_mat[0:3,3]=trans_vec
        tfm_mat[3,3]=1
        print tfm_mat
        # finding new Q w.r.to transformation matrix 'tfm_mat'
        Q=[]
        for loop in range(len(p_cl2)):
            tmp=np.dot(tfm_mat,[p_cl2[loop,0],p_cl2[loop,1],p_cl2[loop,2],
            p_cl2_tmp=tmp[0:3]/float(tmp[3])
            Q.append(p_cl2_tmp)
        p_cl2=np.asarray(Q)


    # Final Q (i.e) transformed point cloud 2 is given by
    Q_trans=p_cl2
    return Q_trans
#########################################
# ICP algorithm ends here
#########################################

#########################################
# Finding correspondence starts here
#########################################

def Corres(p_1,p_2,threshold):

    corres_p=[]
    corres_q=[]
    #p_cl2_dummy
    p_cl2_dummy=p_2
    #idx=None
    dummy_mat=np.zeros((p_1.shape[0],6))
    count=0
    idx_list=[]
    for loop1 in range(len(p_1)):

        diff_sq=(np.subtract(p_1[loop1,:],p_cl2_dummy[:,:])**2)

        euc=np.sqrt(diff_sq[:,0]+diff_sq[:,1]+diff_sq[:,2])

        idx=np.argmin(euc)
        min_val=np.min(euc)
        #print idx
        if idx in idx_list:
            continue
        if threshold>min_val:
            # List cannot be used in python since it uses late binding
            #corres_p.append(p_1[loop1,:])
            #corres_q.append(p_2[idx,:])
            dummy_mat[count,:]=(p_1[loop1,0],p_1[loop1,1],p_1[loop1,2],p_2
            count+=1
            idx_list.append(idx)
        else:
            continue
    idx_list=np.asarray(idx_list)
    for loop in range(len(dummy_mat)):
        if dummy_mat[loop,0]!=0 or dummy_mat[loop,1]!=0 or dummy_mat[loop,2
            corres_p.append(dummy_mat[loop,0:3])
            corres_q.append(dummy_mat[loop,3:6])

    print "after finding correspondence"
    return np.asarray(corres_p),np.asarray(corres_q)

#########################################
# Finding correspondence ends here
#########################################
```

This is the end of the document.