PROBLEM 2

Part A

The function to estimate correspondence can be seen as below:

```
#Function that constructs the list C of estimated point correspondences
def EstimateCorrespondences(X,Y,t,R,dmax):
    C = [] #Initialize empty set of correspondences
    X_TF = ((R@X.T)+t).T #Apply transformation to the X pointset

    #pairwise_distances_argmin_min() Function takes 2 set of data points.
    #Referencing the entries of FIRST LIST and returns the array of
    #shortest distance and index of point that corresponds to it in the second list
    neighbours = pairwise_distances_argmin_min(X_TF, Y)

    #for each correspondence checks the if the shortest distance is < dmax
    for i in range(len(X_TF)):
        if neighbours[1][i]<dmax:
            C.append((i, neighbours[0][i]))
    return C
```

Sample implementation of the above function can be seen below. It returns a list of correspondences whose length (3299) is less than the number data points in point clouds (5750), because initially not every point in X data set is as close to the its nearest point in the Y data set as per the dmax criteria.

```
In [13]: CC=EstimateCorrespondences(X,Y,t0,R0,dmax)
         len(CC),CC

Out[13]: (3299,
          [(339, 3997),
           (564, 3997),
           (566, 3997),
           (567, 3997),
           (613, 3648),
           (722, 5170),
           (724, 5170),
           (734, 3997),
           (736, 3997),
           (738, 3648),
           (739, 3997),
           (740, 3648),
           (778, 3997),
           (779, 3997),
           (803, 900),
           (869, 900),
           (878, 2635),
           (883, 4398),
```

Part B

The function to compute Optimal Rigid transformation can be seen as below:

```
def ComputeOptimalRigidRegistration(X,Y,C):

    #Number of points in correspondences/number of points associated with eachother under dist<dmax
    K = len(C)

    #Create a list of X and Y point clouds that are included in correspondence C:
    Xassociated = []
    Yassociated = []
    for i, j in C:
        x = X[i]
        y = Y[j]
        Xassociated.append(x)
        Yassociated.append(y)

    #Calculate the point cloud centroids:
    x_centroid = sum(Xassociated)/K
    y_centroid = sum(Yassociated)/K

    #Calculate deviations of each point from the centroid of its pointcloud:
    #Xcentered  = Xassociated - x_centroid
    #Ycentered  = Yassociated - y_centroid

    #Calculate deviations of each point from the centroid of its pointcloud:
    Xcentered = X-x_centroid
    Ycentered = Y-y_centroid

    #Compute cross-covariance matrix W:
    W = np.zeros((3, 3))

    for i,j in C:
        x = Xcentered[i]
        y = Ycentered[j]
        W += y.reshape(3,1) @ x.reshape(1,3)

    #Compute singular value decomposition: W = UΣV'
    U, S, V_T = np.linalg.svd(W/K)

    #Construct optimal rotation:
    R = U@V_T

    #Recover optimal translation:
    t = y_centroid.reshape(3,1) - R@x_centroid.reshape(3,1)

    return R,t
```

A sample implementation of the function can be seen as below that returns a rotation matrix and translation. However, it is just for the 1 iteration:

```
In [30]: ComputeOptimalRigidRegistration(X,Y,CC)

Out[30]: (array([[ 0.99439196,  0.07890639,  0.07041592],
                 [-0.0802859 ,  0.99662731,  0.01697608],
                 [-0.0688389 , -0.02253428,  0.99737326]]),
          array([[0.08844609],
                 [0.01414317],
                 [0.07167098]]))
```

Part C

The function for Iterative closest point algorithm can be seen below that runs par(a) and part(b) for **num_ICP_iters** times:

```
# ICP function
def ICP(X,Y,t0,R0,dmax,num_ICP_iters):
    #initialization
    t=t0
    R=R0
    for i in range (num_ICP_iters):
        C = EstimateCorrespondences(X,Y,t,R,dmax)
        R,t = ComputeOptimalRigidRegistration(X,Y,C)
    return t,R,C
```

A sample implementation of the above function can be seen below, where it was implemented for 1 iteration and so the results match the implementation in part A and part B:

```
In [31]: ICP(X,Y,t0,R0,dmax,1)

Out[31]: (array([[0.08844609],
                 [0.01414317],
                 [0.07167098]]),
          array([[ 0.99439196,  0.07890639,  0.07041592],
                 [-0.0802859 ,  0.99662731,  0.01697608],
                 [-0.0688389 , -0.02253428,  0.99737326]]),
          [(339, 3997),
           (564, 3997),
           (566, 3997),
           (567, 3997),
           (613, 3648),
           (722, 5170),
           (724, 5170),
           (734, 3997),
           (736, 3997),
           (738, 3648),
           (739, 3997),
           (740, 3648),
           (778, 3997),
```

PART D:

The task was implemented using the following code:

```
# Creating the numpy array of the point clouds from the provided txt file
X = np.loadtxt(r"C:\Users\AVISH\Downloads\pclX.txt")
Y = np.loadtxt(r"C:\Users\AVISH\Downloads\pclY.txt")

#defining parameters
t0 = np.zeros((3,1))
R0 = np.eye(3)
dmax = 0.25
num_ICP_iters = 30

#Run ICP for num_ICP_iters times
tf,Rf,Cf=ICP(X,Y,t0,R0,dmax,num_ICP_iters)
```

The Results of the following part are as follows:

• The parameters (t, R) ∈ SE(3) for the estimated rigid transformation

```
In [6]: tf,Rf

Out[6]: (array([[ 0.49661487],
                [-0.29392971],
                [ 0.29645004]]),
         array([[ 0.95126601, -0.15043058, -0.26919069],
                [ 0.22323628,  0.9381636 ,  0.26460276],
                [ 0.21274056, -0.31180074,  0.92602471]]))
```

• The RMSE (8) for the estimated point correspondences

```
In [7]: #Finding RMSE
        distance_list = []
        XX=((Rf@X.T)+tf).T
        for i,j in Cf:
            distance=np.linalg.norm(Y[j]-XX[i])
            distance_list.append(distance**2)
        RMSE=math.sqrt(sum(distance_list)/len(distance_list))
        RMSE

Out[7]: 0.008950576587683131
```

• The plot showing the co-registered point clouds

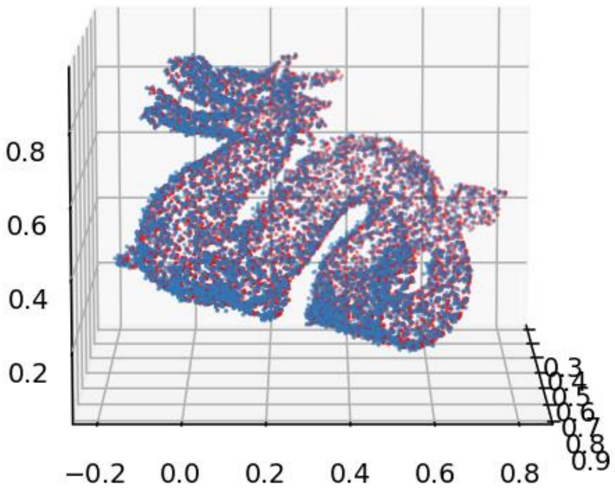

Figure 1: Y [BLUE] and X (after transformation) [RED] plotted together

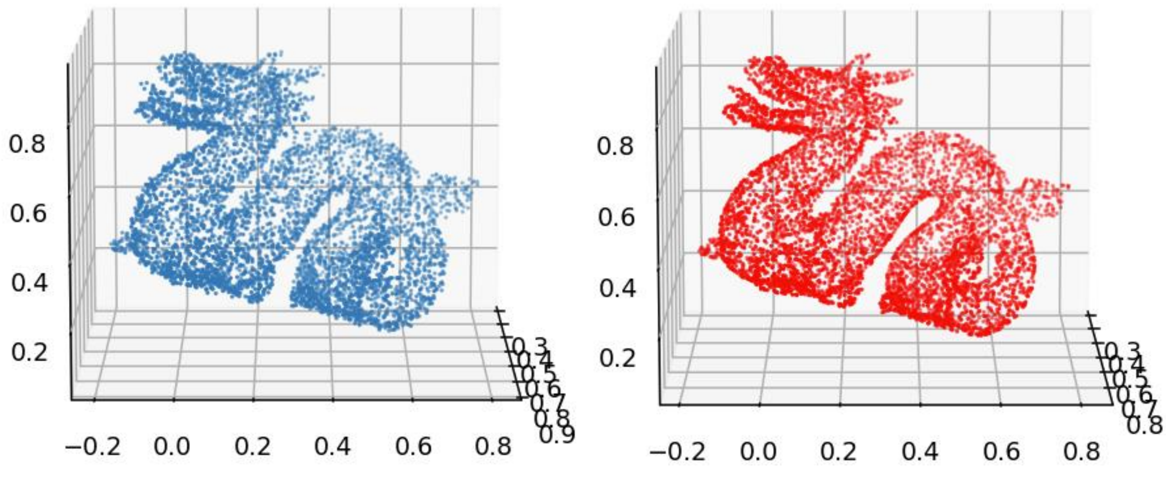

Figure 2: Y data point (LEFT) and X data points after transformation (RIGHT) plotted separately.

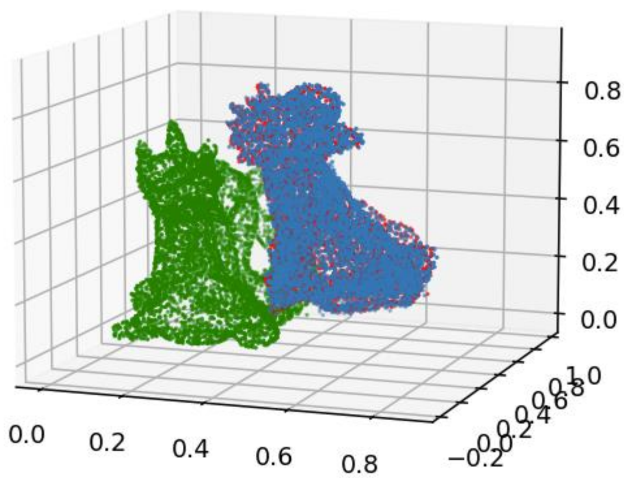

Figure 3: The X data points [GREEN], BEFORE transformation, has shifted and rotated to give X_final [RED] after transformation to meet the reference points Y [BLUE].

- Your code:

NOTE:

The code can be found in the attached ICP.ipynb file