TP 2: Iterative Closest Points algorithm for point cloud registration

Objectives

- Test ICP algorithm on Cloud Compare
- Implement your own ICP and analyse its behaviour

The report should be a pdf containing the answers to the **Questions** and named "TPX_LASTNAME.pdf". Your code should be in a zip file named "TPX_LASTNAME.zip". You can do the report as a pair, just state both your names inside the report and in the pdf and zip filenames, like "TPX_LASTNAME1_LASTNAME2.pdf"
Send your code along with the report to the email mva.npm3d@gmail.com. The object of the mail must be "[NPM3D] TPX LASTNAME" or "[NPM3D] TPX LASTNAME1 LASTNAME2" if you are a pair working on the report.

A. CloudCompare ICP

In the first part of the practical session, you will use the registration tool in Cloud Compare software to visualize a point cloud registration.

1) Load bunny_original.ply and bunny_perturbed.ply artificial clouds in Cloud Compare. Select both clouds and start ICP: "Tools > Registration > Fine Registration (ICP)". You can use the default parameters.

2) Load bunny_original.ply and bunny_returned.ply artificial clouds in Cloud Compare and try to use ICP on those clouds.

3) Load Notre_Dame_Des_Champs_1.ply and Notre_Dame_Des_Champs_2.ply urban 3D scans from a mobile mapping system and try to use ICP on those clouds.

Question 1: How well does ICP perform on those examples? What is the difference between the aligned cloud and the reference cloud? In the last example (Notre Dame), which cloud should be the reference and why?

B. Rigid transformation between matched set of points

The ICP algorithm relies on finding the best transformation at each step. It is defined as the transformation minimizing the distances between matched points. Finding such a transformation is not only useful for ICP: when using targets to align two point clouds, you have two set of matched points and need the transformation aligning them.

Let's call P the reference points and P' the points we need to align with P. Both P and P' are $(d \times n)$ matrices where n is the number of points and d the dimension. Our goal is to find the rotation matrix R and the translation vector T minimizing:

$$\sum_{i=1}^{n} \|p_i - (R \times p_i' + T)\|^2$$

As a reminder here are the steps to do it:

- Calculate barycenters p_m and p_m'
- Compute centered clouds Q and Q'
- Get matrix $H = Q' \times Q^T$
- Find the singular value decomposition USV^T of H
- Return $R = VU^T$ and $T = p_m - Rp_m'$

This method is quite simple but have a flaw: it only ensure that the matrix R is orthogonal. As we don't want R to be a reflexion, but only a rotation, we have to make sure that $det(R) = 1$. This can be done by a simple trick: in the case $det(R) < 0$, multiply the last column of U by -1 and compute $R = VU^T$ again.

1) In ICP.py, implement the function best_rigid_transform with the SVD method described above.
   Tip: you can use the function np.linalg.svd to find a singular value decomposition.

2) To verify your implementation, apply your best rigid transformation function to align bunny_original.ply and bunny_returned.ply. Both point clouds have their points in the same order, so you don't need to find any match. Visualize the result on Cloud Compare. To measure the distance between two set of points, we will use the same metric as Cloud Compare: the root mean square error (RMS) defined below. Compute the RMS error between the points of each cloud before and after applying the best rigid transform.

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\|p_i - p_i'\right\|^2}$$

Question 2: Report the RMS errors obtained. Why did the alignment worked while CloudCompare ICP could not align those two clouds? Would this function align the 3D scans of "Notre Dame des Champs"? Why?

C. Point to point ICP

To align two point clouds, without knowing the matches between their points, we have to proceed step by step. This is why the ICP algorithm is iterative. Each iteration of the ICP algorithm is a succession of two operations:

- Match points
- Apply the transformation minimizing the distances between matched points

As you already implemented the transformation, you only need a point matching strategy to code your own ICP. As the name of this algorithm suggest, every points of the data cloud will be matched with its closest neighbor in the reference cloud.

ICP iterations have to stop at a certain point. The most obvious condition is to stop when reaching a certain number of iterations. The algorithm can also be stopped when the RMS error between clouds gets below a threshold.

1) In ICP.py, implement the function icp_point_to_point. As input, this function takes the data cloud, the reference cloud, and the two control parameters max_iter and dist_threshold. It should return the aligned cloud, two lists of transformations (R_list and T_list) containing the successive transformations found by ICP, and the list of neighbor indices in the reference cloud at each iteration
   Tip 1: Use a kd-tree built on the reference cloud to compute the nearest neighbors
   Tip 2: Use the method list.append to add an element to a list

2) Use your ICP implementation to align ref2D.ply and data2D.ply.

3) Read the docstring (little text at the beginning of the function) of the function show_icp in "visu.py". Pay attention to the remark on R_list and T_list. Use this function to visualize the convergence of your ICP on 2D data

4) Use your ICP implementation to align bunny_original.ply and bunny_perturbed.ply. You can try to visualize the convergence with show_icp, however 3D plot in python

are not very beautiful and quite slow.

5) Modify your ICP implementation so that it returns the RMS between matched points at each step

Question 3: **Plot the RMS during ICP convergence for those two examples (2D and bunny).**

Question 4: **Comment on the two previous curves.**

D. Going further (BONUS)

With your current implementation, the two clouds Notre_Dame_Des_Champs_1.ply and Notre_Dame_Des_Champs_2.ply are too big to test ICP. However, a simple trick can help to reduce the computation time: at each iteration, only search the neighbors of a fixed number of points. You can chose those points randomly across the data cloud. Following the model of icp_point_to_point, implement a function icp_point_to_point_fast, which uses this trick. It should have one more parameter named sampling_limit controlling the number of point used at each iteration.

Tip: Numpy provide useful function like np.random.choice to select random indices in a list. Pay attention to the "replace" parameter of this function.

Question Bonus: **Plot the RMS (computed on all points) during ICP convergence for the "Notre Dame Des Champs" clouds with 1000 and 10000 points used at each iteration. What do you think of those curves?**