

TP 2 : Iterative Closest Points algorithm for point cloud registration

NPM3D - January 23th, 2020
mva.npm3d@gmail.com

Objectives

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

The report should be a pdf containing the answers to the Questions and named "TPX_LASTNAME_Firstname.pdf". Your code should be in a zip file named "TPX_LASTNAME_Firstname.zip". You can do the report as a pair, just state both your names inside the report.

Send your code along with the report to the email address above. The object of the mail must be "[MVA_NPM3D] TPX LASTNAME Firstname". If you do the report as pair, the object of the mail must be "[MVA_NPM3D] TPX LASTNAME1 Firstname1 LASTNAME2 Firstname2".

A. CloudCompare ICP

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

- 1) Load [bunny_original.ply](#) and [bunny_perturbed.ply](#) artificial clouds in CloudCompare. 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 CloudCompare and try to use ICP on those clouds.
- 3) Load [Notre_Dame_Des_Champs_1.ply](#) and [Notre_Dame_Des_Champs_2.ply](#) real 3D scans. and try to use ICP on those clouds.

Question 1 (1 points) : 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 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 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 $T = p_m - R p'_m$

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 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 clouds have their points in the same order, so you don't need to find

any match. Visualize the result on CloudCompare. To measure the distance between two set of points, we will use the same metric as CloudCompare : 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.

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

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



If your RMS is not close to zero, ask for teacher's help (or send an email), or you will not be able to answer the next questions.

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 : Build a scikit-learn kd-tree 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 `utils` folder. 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 (2 points) : On two different graphs, plot the RMS during ICP convergence for those two examples (2D and bunny).

- 6) With your current implementation, the two clouds `Notre_Dame_Des_Champs_1.ply` and `Notre_Dame_Des_Champs_2.ply` are too big. 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_stochastic`, 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.

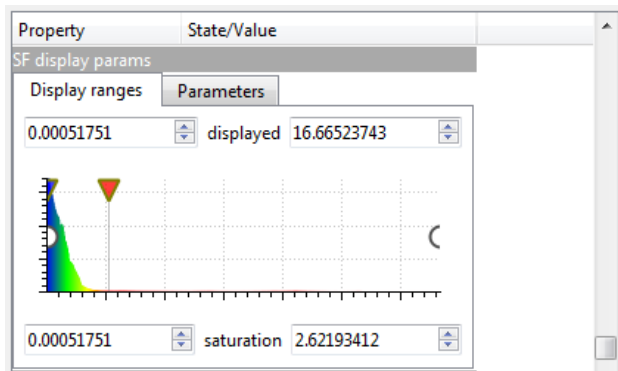
Question 4 (3 points) : On the same graph, plot the RMS during ICP convergence for the "Notre Dame Des Champs" clouds with 1000, 10000 and 50000 points used at each

iteration. What do you think of those three curves?

D. Going further

In the last part of this practical session, you are going to learn tricks to optimize your ICP results and maybe even do better than CloudCompare.

- 1) Load [Notre_Dame_Des_Champs_1.ply](#) and [Notre_Dame_Des_Champs_2.ply](#) in CloudCompare. For each point of Notre_Dame_Des_Champs_2 you can compute the distance to Notre_Dame_Des_Champs_1 with the cloud to cloud distance tool (Tools > Distances > Cloud/Cloud Dist.). This tool will save the distances as a scalar field.



Tip (for a better visualization) : When this is done, select the cloud and find the scalar field display options in the "Properties" panel. Here you can adjust the saturation of the scalar field (slide the red triangle or directly enter a specific value in the saturation box)

Question 5 (2 points) : Do the farthest points (in red color) have a corresponding point in the reference cloud? What effect do they have on ICP convergence? Do you think that computing RMS on the whole data cloud is a relevant measure of the convergence in that case? Why?

(BONUS)

- 2) Now try ICP on those two clouds, but this time, play with the “Final overlap” parameter. Here is what CloudCompare documentation says about it :

***Final overlap:** this is a new parameter for the version 2.6.1. It lets the user specify the actual portion of the data/registered cloud that would actually overlap the model/reference cloud if both clouds were registered. This let the user register entities with only a partial overlap (down to 10% or even less).*

- 3) When using this parameter, CloudCompare RMS becomes irrelevant as a measure of the alignment quality. To be able to compare the results, you have two choices :
 - a) Look at the clouds yourself with CloudCompare and check visually if some object are well aligned (you can look at lamp post, or road signs for example). Using the cloud to cloud distance tool can help the visualization.
 - b) Save the aligned clouds as .ply files (choose binary when asked) and create your own python function to compute a relevant RMS measure between two clouds. Following your answer at question 5, instead of computing RMS on all points, you can compute RMS on the 50%, 70% or 90% closest points, or even all 3 to have more relevant measures.

Question Bonus 1 : With the method of your choice, compare the results of your ICP with the results of CloudCompare ICP (with and without changing final overlap parameter).

- 4) In your [icp_point_to_point_stochastic](#) implementation, add a final_overlap parameter and modify the function following this idea : at each iteration, after choosing a random number of point to compute one step, remove the farthest points depending on the final overlap parameter.
- 5) In the data folder, you can find [Notre_Dame_Des_Champs_2_CCbest.ply](#), this is the best alignment that can be obtained with CloudCompare.

Question Bonus 2 : Compare your best ICP result with the best results obtained by

CloudCompare. Could you beat CloudCompare ICP?