TP 3: Neighborhood descriptors

Objectives

- Compute normals on a point cloud
- Understand different types of local 3D descriptors

A. CloudCompare normals

In the first part of the practical session, you will use the "Normals" tool in CloudCompare software to visualize normals computed on a point cloud.

1) Load Lille_street_small.ply cloud in CloudCompare. Select the cloud and compute the normals: "Edit > Normals > Compute". You can choose "Plane" as the local surface model, and a radius of 50cm for the neighborhoods. Uncheck the orientation box to save computing time.

2) Don't worry if the cloud becomes a bit ugly after this step. It comes from the bad normal orientation. Just select the cloud and in the property panel, uncheck the "Normals" box.

3) Now that you have normals calculated you can visualize them using the tools in "Edit > Normals > Convert to". Convert normals to Dip / Dip direction. It will save dip and dip direction as scalar fields. In the properties panel, you can select the "Dip (degrees)" scalar field, it represents the angle between the normals and the vertical direction.

4) Try this again with smaller and bigger radiuses of neighborhoods. *Tip: If you have a slow computer, do not try a very big value as it would take a very long time (2m should be enough).*

<u>Question 1:</u> If you use a too small / too big radius, what is the effect on the normal estimation? Use screenshots to support your claims.

<u>Question 2:</u> How would you choose the neighborhood scale for a good normal estimation?

B. Local PCA

In this part of the practical session, you will compute principal component analysis (PCA) on neighborhoods, and use the principal components/vectors as local descriptors of the point cloud. If we call a point cloud $\mathcal{C} \subset \mathbb{R}^3$, the definition of the spherical neighborhood of point $\mathbf{p}_0 \in \mathbb{R}^3$ in $\mathcal{C}$ with radius $r \in \mathbb{R}$ is:

$$\mathcal{N}(\mathbf{p}_0, r) = \left\{ \mathbf{p} \in \mathcal{C} \;\middle|\; \|\mathbf{p} - \mathbf{p}_0\| \leqslant r \right\}$$

Computing PCA on this neighborhood simply means finding the eigenvectors and the eigenvalues of its covariance matrix defined by:

$$\text{cov}(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{\mathbf{p} \in \mathcal{N}} (\mathbf{p} - \bar{\mathbf{p}})(\mathbf{p} - \bar{\mathbf{p}})^T$$

Where $\bar{\mathbf{p}}$ is the centroid of the neighborhood $\mathcal{N}$.

1) In descriptors.py, implement the function PCA returning the eigenvalues and the eigenvectors of the covariance matrix.
   Tip 1: This function is very similar to the function best_rigid_transform *(TP2). You can see the similarity between the matrix H and the covariance matrix here.*
   Tip 2: you can use the function np.linalg.eigh *to find the eigenvalues (in ascending order) and the corresponding eigenvectors.*

2) To verify your implementation, apply your PCA to Lille_street_small.ply cloud. You will find the good values as a comment in descriptors.py "main".

You can see a scheme of a PCA on 2D points on figure 1. The first eigenvector follows the direction spreading out the points at most and the first eigenvalue measures that spread. Then the following eigenvectors follow the same rule (spreading out the points at most) with the condition to be perpendicular to all previous eigenvectors. As a convention we will name the eigenvalues λ_1, λ_2 and λ_3 such that $\lambda_1 \geq \lambda_2 \geq \lambda_3$. The corresponding eigenvectors will be named $\boldsymbol{e_1}$, $\boldsymbol{e_2}$ and $\boldsymbol{e_3}$. Be careful with your implementation as the function np.linalg.eigh returns the eigenvalues and eigenvectors in the ascending order, which is the inverse of the convention.

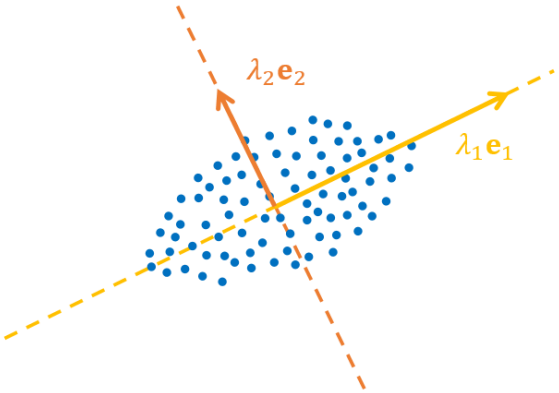

Figure 1 : PCA illustration in 2D

3) If you consider a group of points describing a small part of a 3D surface, which one of the three eigenvectors is expected to be the normal of the surface.

4) In descriptors.py, implement the function compute_local_PCA returning the eigenvalues and eigenvectors of each query point neighborhood in the cloud. This function should take a radius as parameter, and the returned values should be stored in numpy arrays with the shapes [N x 3] and [N x 3 x 3]
Tip: To reduce computation time, you can search for all neighbors at one with the query radius of KD Tree structures. Then loop over neighbors lists to compute local PCAs.

5) Use the result of this function to obtain normals on Lille_street_small.ply cloud with the radius of your choice and save the cloud.
Tip: For the function write_ply to work, your normals should be in a [N x 3] *np array, and you should add the three new field names : "nx", "ny", "nz"*

<u>Question 3:</u> Show a screenshot of your normals converted as "Dip" scalar field in CloudCompare.

<u>Question 4:</u> How could you measure the quality of your normal estimation with the eigenvalues?

6) As explained earlier, "Dip" scalar field represents the angle between the normals and the vertical direction. We can use this angle to compute a simple descriptor that we could name "verticality". It should take values in [0,1] and be close to 1 if the normal is vertical and close to 0 if the normal is horizontal.

$$verticality = \left| 1 - \frac{2\text{angle}(\mathbf{n}, \mathbf{e}_z)}{\pi} \right| = \frac{2 \arcsin(|\langle \mathbf{n}, \mathbf{e}_z \rangle|)}{\pi}$$

The normal verticality is not the only descriptor than you can get with a PCA. Here are three other simple descriptors based on eigenvalues:

$$linearity = 1 - \frac{\lambda_2}{\lambda_1} \quad planarity = \frac{\lambda_2 - \lambda_3}{\lambda_1} \quad sphericity = \frac{\lambda_3}{\lambda_1}$$

7) In descriptors.py, implement the function compute_features returning the 4 features for all query points in the cloud. This function will use compute_local_PCA and thus have a neighborhood radius parameter.
Tip 1: you can use simple operations and more complex function like np.arcsin or np.abs directly on entire arrays. It will be faster than using "for" loops.
Tip 2: if a point has no neighbors, the eigenvalues associated to it will all be equal to zeros. Add a small epsilon to the fractions denominators to avoid errors.

8) Compute those 4 features on Lille_street_small.ply cloud with the radius of your choice and save the cloud.
Tip: the function write_ply can save as many scalar fields as you want with the points.

<u>Question 5:</u> Show screenshots of the 4 features as scalar fields of the cloud. Can you explain briefly the names of the 3 last features considering their definition with eigenvalues?