Python package for lidar point cloud

1 Data extraction:

First we extract data with Pandas and put them in to a numpy array where the points are as [x,y,z]

2 Determining the number of wires :

For this we use AgglomerativeClustering from sklearn.cluster library. It allows us to link points by agglomeration.

The params :

n_clusters=None, we don't know how many wires there are

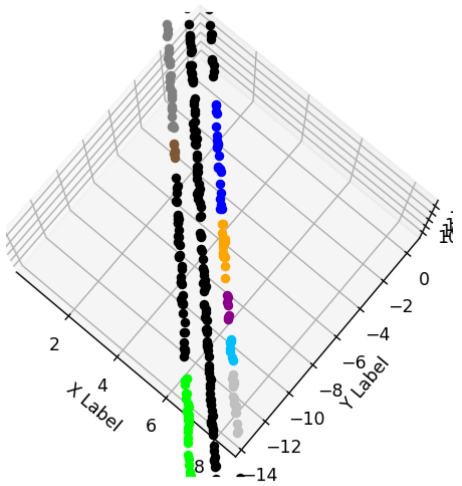
linkage='single', the linkage method to minimize the distance between the points/agglomerations

distance_threshold=0.75 the threshold use to know if points/agglomerations need to be linked

The function cluster makes this with the data given as a numpy array containing points stored as [x,y,z]

The function cluster returns the points classed by agglomerations.

Try to agglomerate with projection on plane x,y and linear regression



We then agglomerateWithPlane use from the planeLib to link two batches which seem to be in the same plane. For this we compare the mean distance between the points from a batch and the plane from the other batch. The mean distance threshold can be changed and adapted with

3 Determining the best fitting plane :

In planeLib

We use PCA from sklearn.decomposition.

It changes the base from 3D to 2D with least loss.

We can also calculate the equation of the plan by reversing base change with 3 points in the plan.
The 3 points are expressed in 3D base but are in the plan. From these 3 points we can determine the equation of the plan.

The function findPlane from planeLib makes all the above with the data given as a numpy array containing points stored as [x,y,z]
The function findPlane returns the coefficient a,b and d of the plane equation as ax+by+d=z
It also returns the initial array after changing the base in which the points inside the array are expressed. So when returned, the points are in 2 dimensions.
Then it returns the PCA object used by sckitlearn to change base.
We call findPlane in the main

4 Finding Catenary Model in 2D:

Everything is in catenarylib, we just call the last function in the main

We first try to find x0 and y0.
For now x0 is fixed at 0. By visualizing data in a plane it showed that x0 seems to be always in 0.
So we keep x0=0 and try to find some points around this coordinate to determine y0 (the mean y from those points)
So we set x0 and y0 which can be upgraded in the future for better precision

We now try to determine the best c
For this we try to minimize the distance between the points from the cloud and the curve from the catenary equation.
We use catenary_dist_function to get the distance between a point and a point from the catenary curve.
We use minimum_distance_point_catanary to find the minimum distance between a point and a point from the catenary curve, so the distance between a point and the curve. (It applies fmin from scipy.optimize to the previously mentioned catenary_dist_function)
We use error_on_catenary to sum the distance between the curve and all the points from the cloud. (applies the previously mentioned minimum_distance_point_catanary to determine the distance between each point and the curve)
Then we use findCatenary to find the parameter c for which error_on_catenary has its minimum value. This returns the value of c as well as the x0 and y0 used.

4 Changing base back to 3D:
In the main
We apply the PCA previously used to change the base again from 2D to 3D

5 Data visualisation:

To help visualization the lib plotlib has been created.

plot

You can plot a simple point cloud by just passing an array containing 3D points to the function plot.

If the cloud is sorted such as after step 2, you can enter the number of different labels with number_of_labels. The points from a same wire will have the same color (up to 15 colors, we could add more manually in the future (variable colors))

plot2D

You can plot 2D point clouds in unicolor with the function plot2D.

You can also set the catenary param to True and enter x0,y0 and c to draw the catenary curve.

6 Future upgrade:

The selection of x0 and y0 (part 4) might be better if we make them as variables and try to minimize the function as we did with c. Yet it may take time (optimizing c takes approximately 50 seconds and there aren't many points in the cloud).

We might upgrade the way we determine the number of wires. It works but isn't enough when there are too big and/or too many shortages of points.

Try to find better thresholds especially for the agglomeration by line

Giving the possibility to get data clouds from a cloud might be a good idea.

The possibility to draw catenary curves in 3D space might help verifying the solutions.