Assignment 5

Processing raw LIDAR data, DBSCAN clustering and documentation of the project on Github.

Author: David Delgado
Course: Industrial AI D7015B 55031
Date: *30-08-2025*

The full code, plots, and README can be found in my GitHub repository:

https://github.com/MANDAVDEL/Assignment_5

Background

LiDAR (Light Detection and Ranging) is a method that uses laser pulses to measure distances to objects and create high-resolution 3D point clouds of the environment (Patwardhan, 2023). Today, point cloud data is used in many applications such as city modeling, environmental monitoring, forest surveys, and infrastructure inspection. For railways and power lines, LiDAR is especially useful to detect ground levels, cables, and support structures (Weinmann, Jutzi, & Mallet, 2014).

Processing LiDAR data requires several steps: first noise is filtered out and the ground surface is identified, then the points are grouped into meaningful objects, and finally the largest or most relevant clusters are analyzed (Meng, Currit, & Zhao, 2010). This assignment aims to go through this process step by step using python and machine learning methods.

Ground level detection

A key step in LiDAR processing is to find the true ground level, since it is often used for further analysis (Patwardhan, 2023). A common method is to use a histogram of the height coordinate (z) to find the most frequent height range, which usually represents the ground (Patwardhan, 2023). This step is very important because ground points can make up to 70% of all measured points and must be separated from objects above the ground (Patwardhan, 2023).

Clustering with DBSCAN

After removing ground points, the structures above ground need to be analyzed. Here we can use DBSCAN (Density-Based Spatial Clustering of Applications with Noise), a popular algorithm for grouping points based on density (Khandelwal, 2020). DBSCAN can find clusters of arbitrary shapes and also handle noise. An important part is to choose the right epsilon value (eps), which sets the maximum distance neighbors points. This is usually done using a k-distance curve, where the "elbow" point shows the optimal value (Machine Learning Knowledge, 2020).

Identifying the largest cluster

The final step is to find the largest cluster. By analyzing the span in x- and y-directions (min and max), this cluster can be separated from the others (Weinmann, Jutzi, & Mallet, 2014). This process is essential to extract information from the point cloud. In some LiDAR applications, Random Forest is also used. It is a supervised learning method that classifies points based on features such as height, intensity, and local geometry. DBSCAN, on the other hand, is an unsupervised method that only groups points by density (Liang, et al, 2013)

Aim of the assignment

Assignment 5 is designed to:

- Give practical experience with basic LiDAR processing in Python.
- Introduce clustering methods.
- Practice documentation and version control using GitHub.

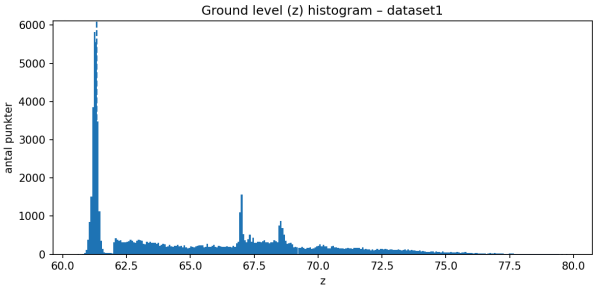
By combining histogram-based ground detection with DBSCAN clustering, I gained a better understanding of the LiDAR data and the workflow used for analysis.

Task 1 Solution in Python:

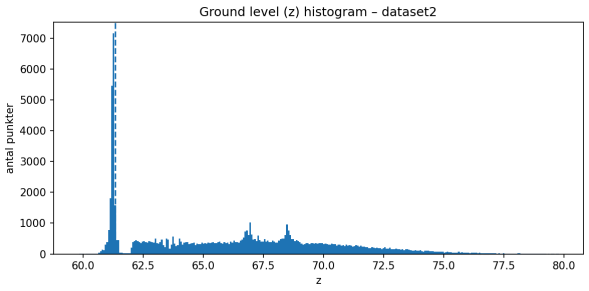
```
'''
Task 1 (3)
- Find the best value for the ground level
- One way: use a histogram (np.histogram)
- Update the function get_ground_level() with your changes

The README should include the ground level for both datasets and the histogram
images.
'''
# %%
import os
import numpy as np
import matplotlib.pyplot as plt

def get_ground_level(pcd, nbins=300, low_prct=20, high_prct=80,
save_plot=False, tag="dataset"):

    """
    Estimates the ground level (z) as the midpoint of the histogram bin with
the most points within the lower part of the height distribution.

    pcd       : Nx3 point cloud (x, y, z)
    nbins     : number of bins in the histogram
    low_prct  : lowest percentile
    high_prct : upper percentile for trimming
    save_plot : if True, save the histogram

    """
    z = pcd[:, 2].astype(float)

    # Focus on lower elevations where the ground is likely located.

    low = np.percentile(z, low_prct)
    high = np.percentile(z, high_prct)
    z_window = z[(z >= low) & (z <= high)]
    if z_window.size == 0:
        z_window = z  # fallback om urvalet blev tomt

    # Histogram -> select the bin with the most points (mode)

    counts, edges = np.histogram(z_window, bins=nbins)
    k = int(np.argmax(counts))
    ground_level = 0.5 * (edges[k] + edges[k + 1])




    # save histogram
```

```
    if save_plot:
        os.makedirs("images", exist_ok=True)
        plt.figure(figsize=(8, 4))
        plt.hist(z, bins=nbins)
        plt.axvline(ground_level, linestyle="--")
        plt.title(f"Ground level (z) histogram – {tag}")
        plt.xlabel("z")
        plt.ylabel("antal punkter")
        plt.tight_layout()
        plt.savefig(f"images/ground_hist_{tag}.png", dpi=150)
        plt.show()

    return float(ground_level)

pcd1 = np.load("C:/Users/davdel0404/Downloads/AI Lund/Assignement
5/Lidar_assignment-1/dataset1.npy")
print("Dataset1 ranges:")
print("x:", np.min(pcd1[:,0]), "→", np.max(pcd1[:,0]))
print("y:", np.min(pcd1[:,1]), "→", np.max(pcd1[:,1]))
print("z:", np.min(pcd1[:,2]), "→", np.max(pcd1[:,2]))
g1 = get_ground_level(pcd1, save_plot=True, tag="dataset1")
print("Dataset1: Ground level =", g1)

pcd2 = np.load("C:/Users/davdel0404/Downloads/AI Lund/Assignement
5/Lidar_assignment-1/dataset2.npy")
print("Dataset2 ranges:")
print("x:", np.min(pcd2[:,0]), "→", np.max(pcd2[:,0]))
print("y:", np.min(pcd2[:,1]), "→", np.max(pcd2[:,1]))
print("z:", np.min(pcd2[:,2]), "→", np.max(pcd2[:,2]))
g2 = get_ground_level(pcd2, save_plot=True, tag="dataset2")
print("Dataset2: Ground level =", g2)
```

Ground level = 61.30881333333333



Ground level = 61.2680366666667



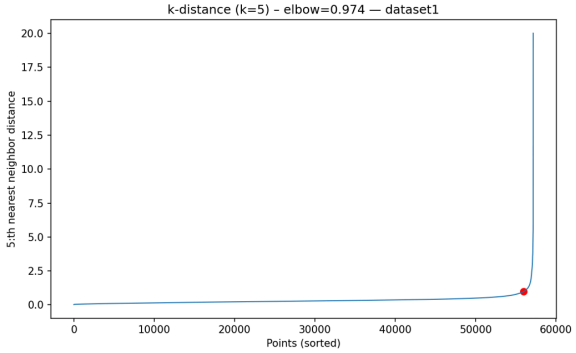
Task 2 Solution in Python:

```
# task2
'''
Find an optimized value for eps.
Plot the elbow and extract the optimal value from the plot
Apply DBSCAN again with the new eps value and confirm visually that clusters
are proper

https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/
https://machinelearningknowledge.ai/tutorial-for-dbscan-clustering-in-python-s
klearn/

For both the datasets
Report the optimal value of eps in the Readme to your github project
Add the elbow plots to your github project Readme
Add the cluster plots to your github project Readme
'''
from pathlib import Path
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import KDTree
from sklearn.cluster import DBSCAN

# Ground levels from task1 used to filter out ground points
GROUND_Z1 = 61.30881333333333
GROUND_Z2 = 61.2680366666667

# Load 3D point cloud datasets
pcd1 = np.load(r"C:\Users\davdel0404\Downloads\AI Lund\Assignement
5\Assignment_5\dataset1.npy")
pcd2 = np.load(r"C:\Users\davdel0404\Downloads\AI Lund\Assignement
5\Assignment_5\dataset1.npy")
# Filter: Keep only objects above ground level
pcd1_above = pcd1[pcd1[:, 2] > GROUND_Z1]
pcd2_above = pcd2[pcd2[:, 2] > GROUND_Z2]

print(f"[Task2] dataset1 above-ground shape: {pcd1_above.shape}")
print(f"[Task2] dataset2 above-ground shape: {pcd2_above.shape}")
```

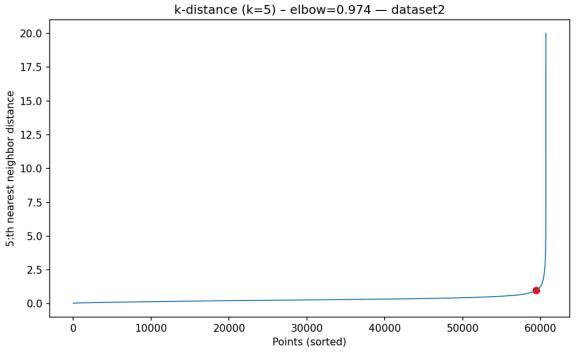
```
def k_distance_elbow(points, k=5, tag="dataset1", save_plot=True):
    #Calculate K distance curve and find the elbow
    #to estimate the optimal epsilon

    points = np.asarray(points)
    tree = KDTree(points)
    dists, _ = tree.query(points, k=k+1)
    kth = np.sort(dists[:, k])

    # Find knee via maximun deviation from the line

    x = np.arange(len(kth), dtype=float)
    y = kth.astype(float)
    x0, y0 = x[0], y[0]
    x1, y1 = x[-1], y[-1]
    num = np.abs((y1 - y0) * x - (x1 - x0) * y + x1*y0 - y1*x0)
    den = np.hypot(y1 - y0, x1 - x0) or 1.0
    idx_opt = int(np.argmax(num / den))
    eps_opt = float(y[idx_opt])

    # Elbow plot: save and show
    if save_plot:
        plt.figure(figsize=(8, 5))
        plt.plot(kth, linewidth=1)
        plt.scatter([idx_opt], [eps_opt], s=40, c="red")
        plt.title(f"k-distance (k={k}) – elbow={eps_opt:.3f} — {tag}")
        plt.xlabel("Points (sorted)")
        plt.ylabel(f"{k}:th nearest neighbor distance")
        plt.tight_layout()
        out = Path("images") / f"kdist_elbow_{tag}.png"
        plt.savefig(out, dpi=150)
        plt.show()
        print(f"[Task2] Sparade elbow-plot: {out}")

    return eps_opt

def run_dbscan_and_plot(points, eps, min_samples=5, tag="dataset1"):
    # Run DBSCAN and save a 2D plot (using the found eps)
    clustering = DBSCAN(eps=eps, min_samples=min_samples).fit(points)
    labels = clustering.labels_
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    print(f"[Task2] {tag}: eps={eps:.3f}, kluster={n_clusters}")

    plt.figure(figsize=(9, 9))
    plt.scatter(points[:, 0], points[:, 1], c=labels, s=2, cmap="Spectral")
    plt.title(f"DBSCAN (eps={eps:.3f}) – {tag}\nKluster: {n_clusters}")
    plt.xlabel("x"); plt.ylabel("y")
    plt.tight_layout()
    out = Path("images") / f"dbscan_clusters_{tag}.png"
    plt.savefig(out, dpi=150)
    plt.show()
    print(f"[Task2] Saved cluster-plot: {out}")
    return n_clusters
```

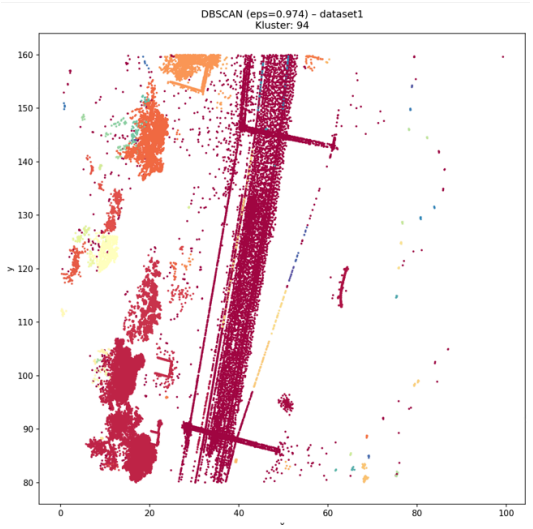
```
# Dataset 1
eps1 = k_distance_elbow(pcd1_above, k=5, tag="dataset1", save_plot=True)
run_dbscan_and_plot(pcd1_above, eps=eps1, min_samples=5, tag="dataset1")

# Dataset 2
eps2 = k_distance_elbow(pcd2_above, k=5, tag="dataset2", save_plot=True)
run_dbscan_and_plot(pcd2_above, eps=eps2, min_samples=5, tag="dataset2")

print(f"[Task2] Optimal eps – dataset1: {eps1:.4f}")
print(f"[Task2] Optimal eps – dataset2: {eps2:.4f}")
```

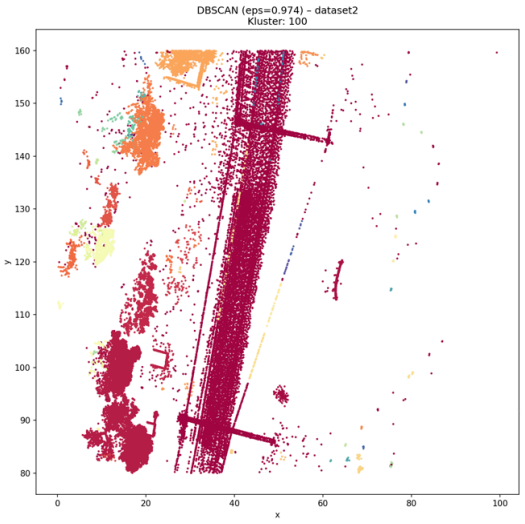
Optimal eps – dataset1: 0.9743



Optimal eps – dataset2: 0.9743







Reference list

Khandelwal, D. (2020, 3 september). *How to master the popular DBSCAN clustering algorithm for machine learning*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/

Liang, J., Zhang, J., Deng, K., Liu, Z., & Zhi, Q. (2013). Point-based classification of power line corridor scene using random forests. *Photogrammetric Engineering & Remote Sensing, 79*(9), 821–833. https://doi.org/10.14358/PERS.79.9.821

Machine Learning Knowledge. (2020, June 8). *Tutorial for DBSCAN clustering in Python Sklearn*. MLK. https://machinelearningknowledge.ai/tutorial-for-dbscan-clustering-in-python-sklearn/

Meng, X., Currit, N., & Zhao, K. (2010). Ground filtering algorithms for airborne LiDAR data: A review of critical issues. *Remote Sensing, 2*(3), 833–860. https://doi.org/10.3390/rs2030833

Patwardhan, A. (2023, 3 mars). *AI factory for railways (AIF/R): Spatial data analytics* [PowerPoint-presentation]. Luleå tekniska universitet.

Weinmann, M., Jutzi, B., & Mallet, C. (2014, September). *Semantic 3D scene interpretation: A framework combining optimal neighborhood size selection with relevant features*. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, II-3*, 181–188. https://doi.org/10.5194/isprsannals-II-3-181-2014