



Object detection with the DBScan algorithm
Programming Exercise P1
(10%)

CSI2110 Data Structures and Algorithms

Fall 2022

This assignment to be done individually.

Programming P1 Nov 1 before 23:55

Late penalty : -30% for late from 1 minute to 24 hs.

Problem Description

In this programming assignment, you will implement a data clustering algorithm named DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Given a large set of data points in a space of arbitrary dimension and given a distance metric, this algorithm can discover clusters of different shapes and sizes, marking as outliers isolated points in low-density regions (i.e. points whose nearest neighbors are too far away).

The DBSCAN algorithm uses two parameters:

- **minPts:** The minimum number of points (a threshold) in the neighborhood of a point for this one to be considered to belong to a dense region.
- **eps (ϵ):** A distance measure that is used to identify the points in the neighborhood of any point.

DBSCAN Algo:

```

// DB contains the list of all points
// label(P) of all P is initialized to undefined
DBSCAN(DB, eps, minPts) {
    C := 0                                /* Cluster counter */
    for each point P in Sequence DB {

        if label(P) ≠ undefined then continue    /* Already processed */
        Neighbors N := RangeQuery(DB, P, eps)    /* Find neighbors */
        if |N| < minPts then {                    /* Density check */
            label(P) := Noise                    /* Label as Noise */
            continue
        }

        C := C + 1                                /* next cluster label */
        label(P) := C                            /* Label initial point */
        Stack S.push{N}                          /* Neighbors to expand */
        while not S.empty() {
            Q = S.pop()                          /* Process point Q */
            if label(Q) = Noise then label(Q) := C /* Noise becomes border pt */
            if label(Q) ≠ undefined then continue /* Previously processed */
            label(Q) := C                        /* Label neighbor */
            Neighbors N := RangeQuery(DB, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                /* Density check */
                S.push{N}                        /* Add neighbors to stack */
            }
        }
    }
}

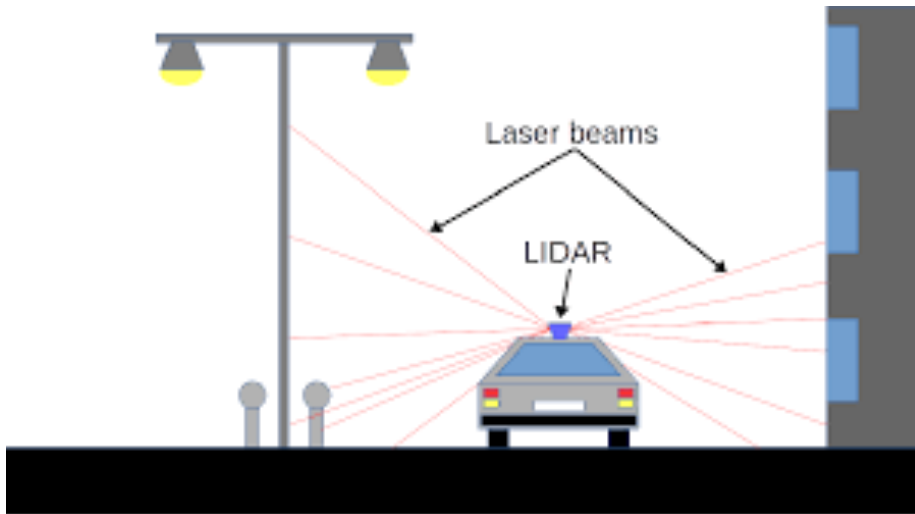
RangeQuery(DB, Q, eps) {
    Sequence N := empty list
    for each point P in Sequence DB {
        if distance(Q, P) ≤ eps then {
            N.add(P)
        }
    }
    return N
}

/* NOTE: S.push{N} means push all elements of list N into stack S */
/* Reference: https://en.wikipedia.org/wiki/DBSCAN */

```

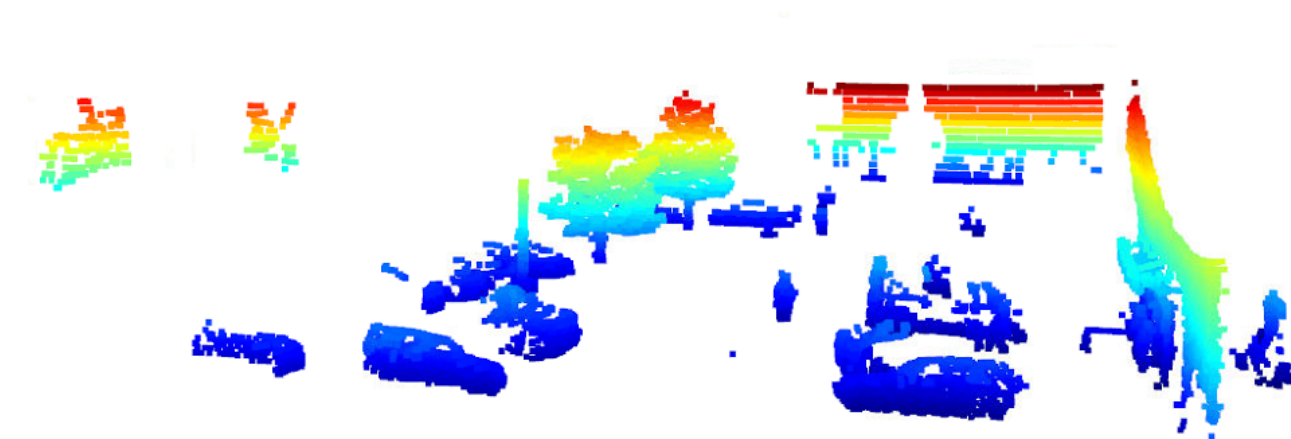
Problem to solve

The intelligent vehicles of the future will be equipped with a multitude of sensors in order to capture information about the surrounding scene and thus being able to autonomously navigate. One of these sensors is the Laser Scanner or LiDAR (Light Detection And Ranging). Using a LiDAR, the vehicle can scan the scene in front by sweeping few laser beams (typically between 8 to 64 lasers).



<https://www.semanticscholar.org/paper/Ego-vehicle-localisation-using-LIDAR-and-compressed-Aronsson-Eriksson/010d3f269728a76ef62ead440541bc9481bc4a58>

Each time a laser beam hit an object, the laser light bounce back to the LiDAR from which a precise distance can be estimated. A complete scan of the scene with these lasers will therefore generate a large set of 3D points (also called point cloud) that correspond to the structure of the scene. The figure on the next pageshows a typical point cloud captured by a car equipped with a LiDAR; note that to facilitate the visualization, the points are color-coded based on their height with respect to the ground. A view of the same scene captured by a camera is also shown.





As it can be seen, each object of the scene will be represented by several 3D points. These object's points should form a cluster. The objective of Part 1 of your programming assignment is therefore to run the DBSCAN algorithm on a LiDAR point cloud in order to detect the objects around the vehicle.

Your task

You will receive 3 datasets, each containing the LiDAR point cloud for a particular scene. The points are listed in a CSV file, one point per line.

```
x, y, z
-5.057771786205823, -4.132708931956775, 0.2428091883181846
-5.0355177525576, -4.088825974461278, 0.241137471769056
-5.125807925277937, -4.136111826946676, 0.2448524059120176
-5.079222136014556, -4.072855314440647, 0.2420290529642492
```

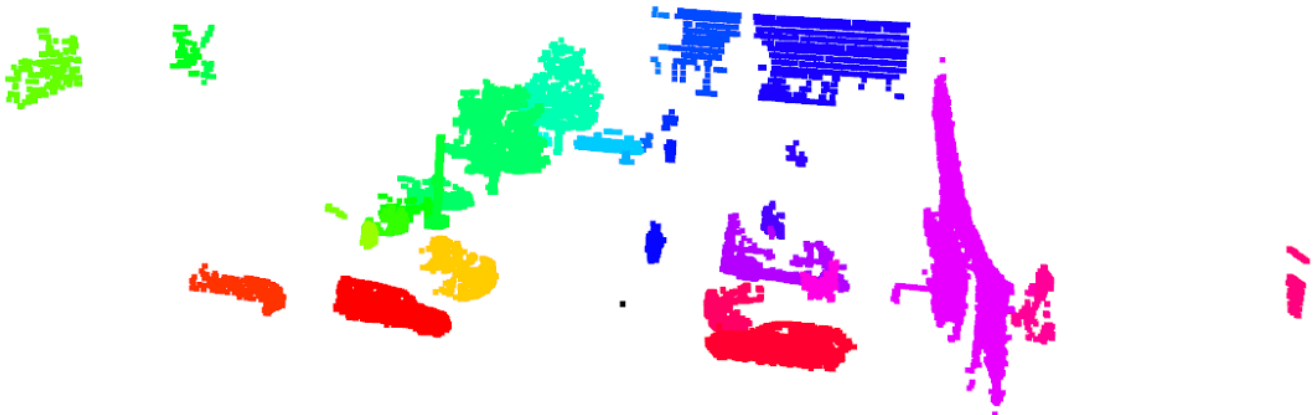
...

We ask you to use the DBSCAN algorithm in order to cluster the points of a LiDAR frame. At the end of the process, each cluster should correspond to a scene object. You have to select the specific parameter values of the DBSCAN algorithm that seems to produce the best results; but you have to use the same parameters for the three datasets.

Your program takes as input the dataset filename and the value of the two DB-Scan parameters (eps, MinPts). As output, it produces a CSV file that will contain the point coordinates and corresponding cluster labels. An RGB color value, unique for each cluster, is also associated to each point; this color will be used to display the resulting clustered point cloud. Each RGB value is a number between 0.0 and 1.0. A display tool that reads this output file will be provided to you.

```
x, y, z, C, R, G, B
-5.057771786205823, -4.132708931956775, 0.2428091883181846, 1, 1.0, 0.0, 0.0
-5.0355177525576, -4.088825974461278, 0.241137471769056, 1, 1.0, 0.0, 0.0
-5.125807925277937, -4.136111826946676, 0.2448524059120176, 1, 1.0, 0.0, 0.0
-5.079222136014556, -4.072855314440647, 0.2420290529642492, 2, 1.0, 0.5, 0.0
```

...



Programming

Your solution must include the following Java classes, attributes and methods:

- The Point3D class that includes
 - getX, getY and getZ methods returning double
 - `public double getX()`
 - `public double getY()`
 - `public double getZ()`
 - a cluster label
 - `int`
 - a distance method that computes the Euclidean distance between two points
 - `public double distance(Point3D pt)`
- The DBScan class that includes
 - a constructor that accepts a List of Point3D
 - `public DBScan(List<Point3D>)`
 - setEps and setMinPts methods
 - `public double setEps(double eps)`
 - `public double setMinPts(double minPts)`
 - the findClusters method that executes the DBScan algorithm
 - `public void findClusters()`
 - a getNumberOfClusters method
 - `public int getNumberOfClusters()`
 - a getPoints that returns the list of Point3D
 - `public List<Point3D> getPoints()`

- the read static method that accept a filename and returns a list of Point3D
 - `public static List<Point3D> read(String filename)`
- the save method that saves all the points with their cluster label and associated RGB color
 - `public void save(String filename)`
- Note that inside the DBScan algorithm, you must use a stack as indicated in the algorithm description
- A Stack class (to be used inside the DBScan algorithm)
 - with push and pop methods
 - you can use your implementation or the one in the Java API
- The NearestNeighbors class that includes
 - a constructor that accepts a List of Point3D
 - `NearestNeighbors(List<Point3D>)`
 - a rangeQuery method that finds the nearest neighbors of a 3D point
 - `public List<Point3D> rangeQuery(double eps, Point3D P)`

You can also create other classes and methods.

Warning: You will find on the Web, numerous implementations of the DBSCAN algorithm. You cannot use them! Do not even look at them! You have to program yourself the algorithm from scratch; not doing so will be considered **plagiarism**.

The name of the output file must be the same than the one of the input file but to which you append the word 'clusters', the value of the parameters used and the number of clusters found using the format filename_clusters_eps_minPts_nclusters. For example, with the input file data.csv, the output file could be:

`data_clusters_1.2_10_57.csv`

Finally, running your program must be done by specifying the input filename, the parameters eps and minpts, exactly as follows:

`java DBScan data.csv 1.2 10`

The parameter values shown are just an example, you can use other values. Before exiting, the program will save the output file and will display to the console the size of all clusters found, from the largest one to the smallest one. And the last value displayed is the number of noise points found.

Marking Scheme

Correctness of the solution:	20%
Quality of programming (structures, organisation, etc)	20%
Quality of documentation (comments and headers)	20%
Adherence to instructions (file format, required methods):	20%
Abstract data types and classes used:	20%

All your files must include a header that includes your name and student number. All the files must be submitted in a single zip file.