

## Object detection with the DBScan algorithm Détection d'objets avec l'algorithme DBScan Exercice de programmation

### Partie 1

(10%)

### CSI2510 Algorithmes et Structure de Données

Automne 2022

*Ce projet est à réaliser de façon individuelle*

**Partie 1 due le 1er Novembre avant 23:59**

*Pénalité de retard : -30% pour un retard de moins de 24 heures*

#### Description du problème

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 ( $\epsilon$ ):** A distance measure that is used to identify the points in the neighborhood of any point.

Cet exercice de programmation vous demande de programmer un algorithme de groupement de données appelé DBSCAN. Soit un ensemble de points dans un espace de dimension quelconque et soit une métrique de distance, cet algorithme permet de trouver les groupements de points de dimension et formes diverses tout en identifiant les points isolés n'appartenant à aucun groupement (i.e. les points pour lesquels leurs plus proches voisins sont éloignés).

L'algorithme DBSCAN utilise deux paramètres :

- **minPts:** le nombre minimum de voisins qu'un point doit avoir pour être considéré être dans une région dense.
- **eps ( $\epsilon$ ):** la distance permettant d'identifier les points appartenant au voisinage d'un point donné.

### DBSCAN Algo:

```

// DB contains the list of all points
// label(P) of all P is initialized to undefined
DBSCAN(DB, eps, minPts) {
    C := 0
    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
    label(P) := C
    Stack S.push{N}
    while not S.empty() {
        Q = S.pop()
        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 /* Scan all points in DB */
            N.add(P) /* Compute distance */
    }
    return N /* Add to result */
}

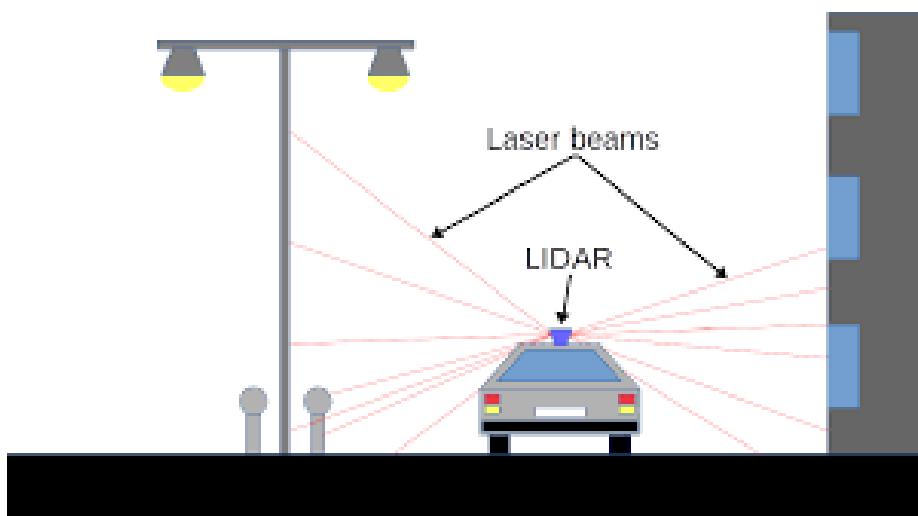
/* NOTE: S.push{N} means push all elements of list N into stack S */
/* Reference: https://en.wikipedia.org/wiki/DBSCAN */

```

### **Problème à résoudre**

*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).*

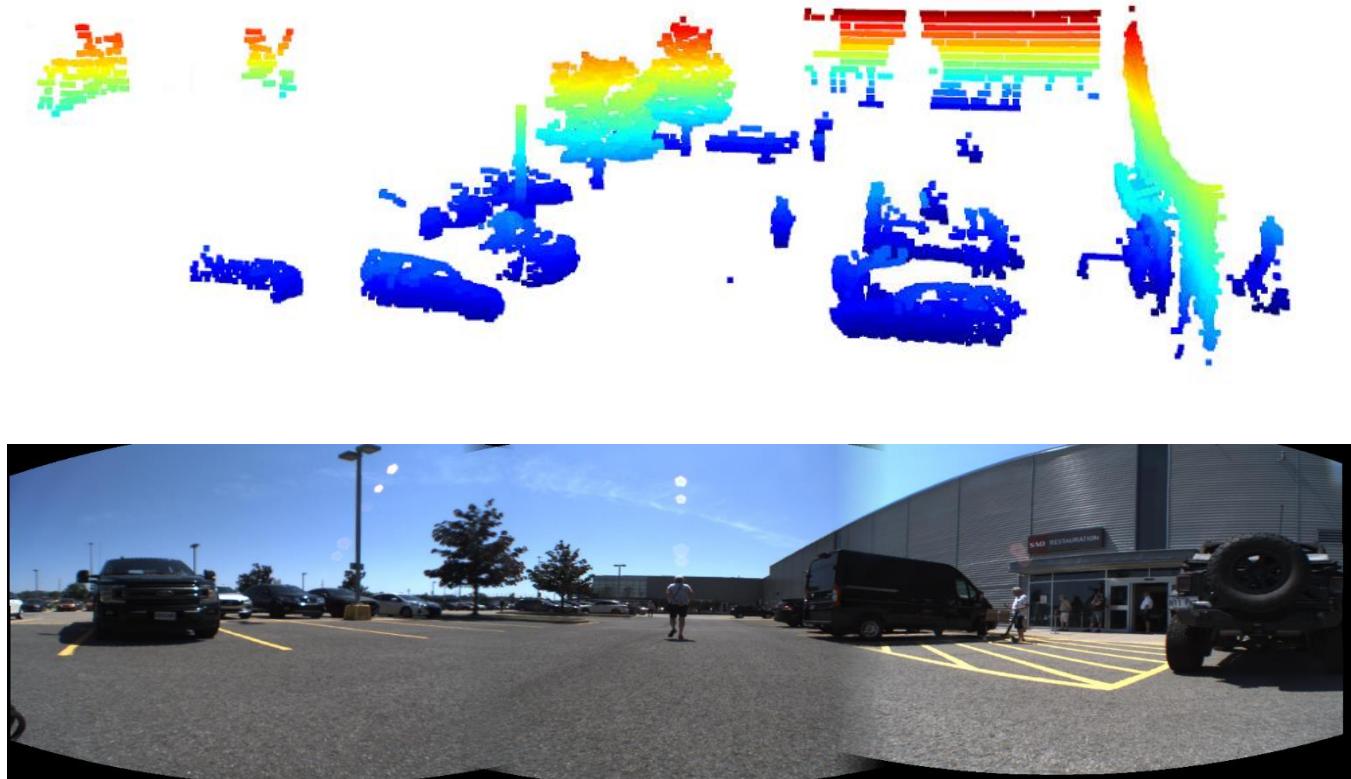
Les véhicules intelligents du futur seront équipés d'une multitude de senseurs afin de capter l'information sur la scène qui les entourent de façon à pouvoir naviguer de manière autonome. L'un de ces senseurs est le Scanneur Laser, aussi appelé LiDAR. En utilisant un tel LiDAR, le véhicule peut examiner la scène devant lui par balayage de faisceaux laser (typiquement entre 8 et 64).



<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 pages shows 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.*

Chaque fois qu'un faisceau laser atteint un objet, la lumière laser rebondit et retourne vers le LiDAR; celui-ci peut alors estimer la distance à cet objet. Un balayage complet de la scène avec des faisceaux laser génère donc un grand ensemble de points 3D (ou nuage de points) représentant la structure de la scène. La figure de la page suivante montre un tel nuage de points tel que capté par un véhicule équipé d'un LiDAR; à notre qu'afin de faciliter la visualisation, les points utilisent un code de couleur basé sur la hauteur des points par rapport à la route. Une image de cette même scène telle que captée par une caméra est aussi montrée.



*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.*

Tel que vous pouvez le constater, chaque objet de la scène est représenté par plusieurs points 3D. Ces points appartenant à un objet devraient normalement créer un groupe. L'objectif de la Partie 1 de votre projet de programmation consiste donc à utiliser l'algorithme de DBSCAN afin d'identifier les objets autour du véhicule.

### **Votre tâche**

*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
```

...

Vous recevrez trois fichiers de données. Chacun de ces fichiers correspond à une capture LiDAR (une trame). Les points de cette trame sont listés dans un fichier CSV, avec un point par ligne.

```
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
```

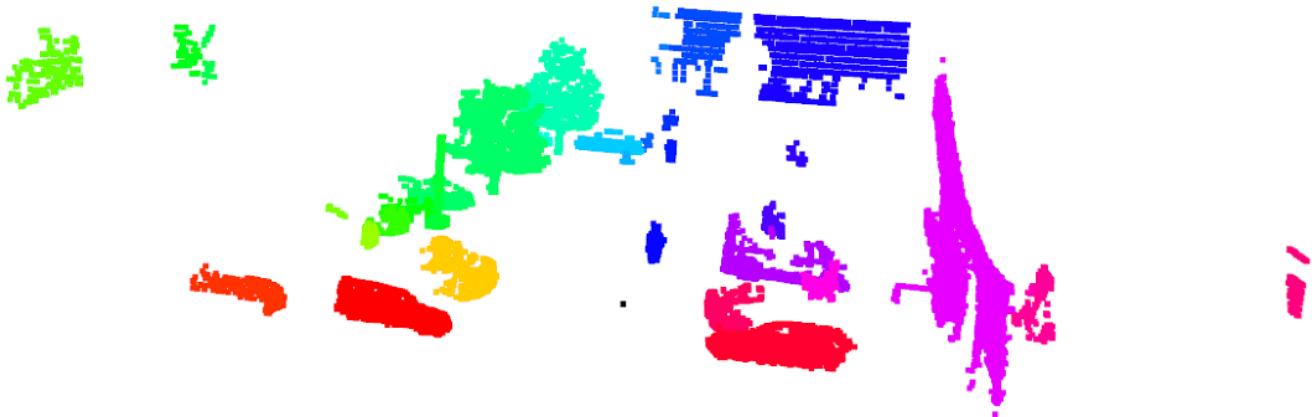
...

Nous vous demandons donc d'utiliser l'algorithme DBSCAN afin de grouper les points d'une trame LiDAR. A la fin du processus, chacun des groupes de points devraient correspondre à un objet de la scène. Vous devrez choisir la valeur des paramètres de l'algorithme DBSCAN qui semble produire les meilleurs résultats; mais vous devez utiliser les mêmes paramètres pour les trois ensembles de données à traiter.

Votre programme prendra en entrée le nom du fichier contenant les points ainsi que la valeur des deux paramètres de l'algorithme DBSCAN à utiliser. En sortie, votre programme doit produire un fichier CSV contenant la liste des points et l'étiquette du groupe auquel chaque point appartient. De plus, une couleur RGB sera associée à chaque étiquette; cette couleur sera utilisée afin de visualiser les points et leur groupe. Chaque valeur RGB est un nombre entre 0.0 et 1.0. Un outil de visualisation de nuage de points vous sera fourni.

```
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
```

...



## Programmation

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)`

*You can also create other classes and methods.*

*Vous pouvez aussi créer d'autres classes ou méthodes.*

***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.*

**Avertissement :** Vous trouverez sur le Web de nombreuses versions de l'algorithme DBSCAN. Vous ne pouvez pas utiliser celles-ci! Ne les regardez même pas! Vous devez programmer vous-même l'algorithme à partir de rien. Utiliser ou s'inspirer d'une source externe sera considéré comme du **plagiat**.

Le nom du fichier de sortie doit être le même que celui du fichier d'entrée mais en y ajoutant le mot 'clusters' et la valeur des paramètres utilisé ainsi que le nombre de groupes trouvée, en utilisant la convention `filename_clusters_eps_minPts_nclusters`. Par exemple pour le fichier `data.csv`, vous pourriez produire le fichier :

`data_clusters_1.2_10_57.csv`

Finalement, votre programme doit se lancer en spécifiant le nom du fichier de sortie ainsi que la valeur des paramètres `eps` et `minpts`. Ceci doit se faire exactement comme suit :

`java DBScan data.csv 1.2 10`

Les valeurs de paramètres montrés sont simplement à titre d'exemple, vous pouvez utiliser d'autres valeurs. Avant de se terminer, votre programme sauvegarde le fichier de sortie et affiche à la console la taille des groupes trouvés du plus grand au plus petit. Et la dernière valeur affichée est le nombre de points 'outliers' trouvés.

### **Grille de correction**

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.*

Tous vos fichiers doivent inclure un entête incluant votre nom et numéro d'étudiant. Tous ces fichiers doivent être soumis dans un fichier zip.