Automated Segmentation and Annotation of 3D Point Clouds for Plant Phenotyping

Phuong Thu Ky Doan

September 2024

1 Introduction

The project is conducted during my internship at the Lincoln Centre for Autonomous Systems (LCAS) in 2024, spanning a duration of three months (between 01/07/2024 and 30/09/2024). The project focuses on developing automated segmentation and annotation systems for strawberry plants 3D point clouds. The motivation behind this project is the vital role of meaningful and accurate information from 3d point clouds in phenotyping research, but manual segmentation of large point cloud datasets is time-consuming. Therefore, an automated system is necessary for reducing human efforts, speeding up the process, and improving the accuracy.

The following tasks are accomplished during the project and are detailed in this report:

- Point cloud data collecting simulation on real tomato plants in polytunnels.
- Color-based segmentation using K-Means.
- Density-based segmentation using DBSCAN.
- Combination of both density and color attributes.

The dataset used in this project is provided on LAST-Straw Project website, consisting of 84 XYZ files with data of 6 strawberry plants.

2 Methodology

"Point cloud is a collection of 3D points that depicts the surface of a 3D scene" (Huang et al., 2023). Typically, point clouds contain information about spatial position of each point, represented by XYZ coordinates, and additional information, such as colours, which further describes the object. The generation of 3D point cloud data starts with scanning a series of locations, with overlap, ensuring all angles of a mapped area are covered. Afterwards, the individual datasets merge to create one accurate point cloud. Clustering is the work of grouping similar data objects into the same class, and cluster is the collection of data objects (Deng, 2020).

2.1 Data Collection and Preprocessing

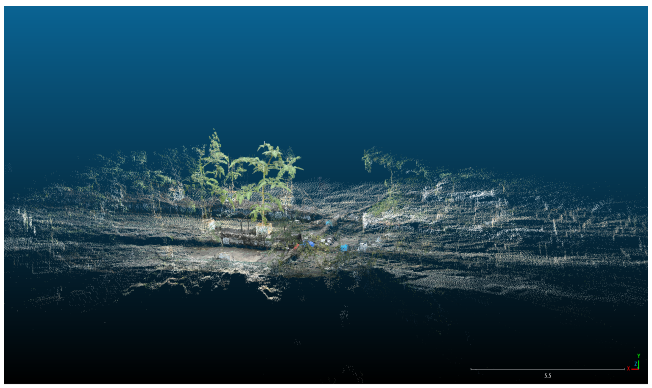
The dataset, which is available on LAST-Straw webpage, consists of 84 point clouds of 6 strawberry plants of 2 varieties, each captured 14 times over 11 weeks (13/05/2022 - 29/07/2022). Of these, 13 are annotated with class and instance labels for each point in the

cloud. The data is saved in .xyz files with six columns of respectively x,y,z spatial coordinates and r,g,b color values of the points, and with annotated files, 2 columns are added for class and instance labels. Seven classes are used to distinguished different plant organs and two additional classes for background information inadvertently captured during the scanning process. There are nine semantic classes:

1. leaf or leaflet
2. stem (incl. petiole, peduncle, pedicel, and stolon)
3. berry
4. flower
5. crown
6. background (incl. soil, grow bags, neighboring plants, and any other objects captured in the background)
7. other (any unidentifiable structure within the plant, which may include newly emerging organs that are not yet identifyable)
8. the scanning table
9. emergent leaf or leaflet

(James et al., 2022)

The demonstrating point cloud data collecting process has been undertaken with two tomato plants in a polytunnel at Riseholme Campus using an Ipad and Luma AI application. The plants have been scanned with a series of locations, then submitted to Luma AI to generate point cloud based on overlap technique. The data cleaning process then has been undertaken using tools provided by CloudCompare. Specifically, noise are removed with 'Cross Section' and 'Noise Filter' tools supported by CloudCompare, coordinate axis is standardised with 'Transformation' tool. Finally, clean point cloud data is exported.



(a) Plant 1.

(b) Plant 2

Figure 1: Visualisation of the original point cloud data from tomato plants in CloudCompare.

(a) Plant 1.

(b) Plant 2

Figure 2: Visualisation of clean point cloud data from tomato plants in CloudCompare.

2.2 K-Means Clustering

K-Means clustering is an unsupervised machine learning algorithm partitioning the dataset into a pre-defined number of clusters. Data points are assigned to one of the clusters based on their distance from the centre of the clusters. The algorithm requires the number of cluster, k, to be specified by the user. The process starts by randomly assign 'k' cluster centroids in the

space. Each point is categorises to its closest centroid, and the centroids are updated to the averages of the items in the clusters. The process iterates for a given number of times to form final clusters.

This approach aimed to perform segmentation based on the similarity in color of the points. Points with close color values are grouped in a same cluster. It is implemented using built-in K-Means method in OpenCV and used color information extracted from the input files as data for processing. The user is prompted to specify the file name and the number of clusters they want their data to be partitioned into. The system iteratively calculates the Euclidean distance between the data points and the clusters' centroids, then assigns them to the cluster with the closest distance to form final clusters.

The approach requires a distance measure method to calculate color difference between points. Distance is calculated on RGB values (columns indexed from 3 to 5) extracted from the input data using Euclidean distance metric. Given two points and their RGB values $P_1(r_1, g_1, b_1)$ and $P_2(r_2, g_2, b_2)$. The formula for Euclidean distance d between them is:

$$d = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

Two variations using two different color spaces were explored in this approach, aiming to examine whether the system can give different results in different color spaces. Besides the original RGB, which is used to represented color information of point clouds in the given dataset, CIELAB, which is well-known for human vision approximation, was chosen. In this variation, after extracting RGB information from the given data, the color space is converted from RGB to CIELAB using built-in method included in Scikit-Image. Then, the CIELAB color information array is used as data for the K-Means clustering algorithm. Similarly, distance is also calculated using Euclidean distance metric. Given two points and their CIELAB values $P_1(L_1^*, a_1^*, b_1^*)$ and $P_2(L_2^*, a_2^*, b_2^*)$. The formula for Euclidean distance d between them is:

$$d = \sqrt{(L_1^* - L_2^*)^2 + (a_1^* - a_2^*)^2 + (b_1^* - b_2^*)^2}$$

In the implementations, color attributes of points were extracted from the data by slicing the array and used to measure the distance with the abover-mentioned formulas.

As prior knowledge of the number of clusters is required in K-Means clustering, the k values chosen for semantic and instance segmentation performance are 9 and the maximum value of instance labels in each ground truth, respectively.

2.3 DBSCAN

Besides the segmentation based on the similarity in color with K-Means, another approach was carried out with DBSCAN, attempting to identify clusters based on the density of points in the data.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised learning method that partitions the dataset into distinctive clusters, based on the idea that a cluster in data space is a contiguous region of high point density separated from the others by contiguous regions of low point density. The algorithm requires two parameters:

1. eps: the radius of neighborhood.

2. minPts: the minimum number of points required to form a dense region.

The algorithm considers each unprocessed point, then checks the neighborhood within an eps distance. If there are more than minPts in the neighborhood, then the query point is

labeled as a core point and a new cluster is formed with it and its neighborhood. It then checks each of the neighbors and add them to the cluster if they are also core points. Otherwise, they are labeled as noise points.

The implementation was done using built-in DBSCAN method in Scikit-learn. With two specified parameters (eps:the radius of neighborhood, minPts: the minimum number of points required to form a dense region), segmentation is performed by calculating pairwise Euclidean distance between a point and the others.

This project has implemented both traditional DBSCAN and a generalised version of it, which allows the consideration of both spatial and color information in assigning a point to its cluster. Specifically, with the traditional version, clusters are formed based on the density of points in space, in which the Euclidean distance of each point to the others is calculated on their x,y,z values. Given two points and their spatial attributes: $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$. The formula for Euclidean distance d between them is:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

In contrast, in the generalised version, the distance between points is redefined with the contribution of both spatial and color information. Points in the same cluster should be close to each other in space, and should also have similarity in color. The distance method in this version is the sum of spatial Euclidean distance and color Euclidean distance, with specifiable coefficients to standardise the importance of the variables. Given two points with their spatial attributes and color attributes: $P_1(x_1, y_1, z_1, r_1, g_1, b_1)$ and $P_2(x_2, y_2, z_2, r_2, g_2, b_2)$. The formula for Euclidean distance d between them is:

$$d = a\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} + b\sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

(with a and b are the specifiable coefficients)

For both two DBSCAN approaches, the values for two required parameters (eps and minPts) are chosen as 10 and 200, respectively.

DBSCAN Algorithm basic steps:

Require: $eps \geq 0$, $minPts > 0$
for each unprocessed point $x \in X$ **do**
  mark x as visited
  $N \leftarrow GetNeighbors(x, eps)$
  if $|N| < minPts$ then
    mark x as a noise point
  else
    $C \leftarrow \{x\}$
    for each neighbor point $x' \in N$ **do**
      $N \leftarrow N \backslash x'$
      if x' is not visited **then**
        mark x' as visited
        $N' \leftarrow GetNeighbors(x', eps)$
        if $|N| \geq minPts$ **then**
          $N \leftarrow N \cup N'$
        end if
      end if
      if x' is not in a cluster yet **then**
        $C \leftarrow C \cup x'$
      end if
    end for
  end if
end for

Table 1: DBSCAN Algorithm Basic Steps

2.4 Data Visualisation

Visualisation method for segmented point cloud data has been prepared with Open3D library in Python. Each class/instance of point cloud objects are visualised with a different color. Technically, a number of color attributes is randomly generated and stored in a 2D array. An iteration runs over the data and partitions it into different lists of classes/instances and replace the original RGB color attributes with the prepared ones. Semantic and instance segmentation visualisation modes can be switch with a hot key.

(a) Semantic Segmentation. (b) Instance Segmentation

Figure 3: Visualisation of an annotated point cloud data with Open3D.

2.5 Evaluation Metrics

In this project, result quality, execution time, and memory consumption of the algorithms are evaluated. Intersection over Union (IoU) metric was used to evaluate the accuracy of the segmentation works using 5 in 13 annotated files provided in LAST-Straw dataset as ground truths (GT). Candidates were extracted from GT and segmented result (RS) files based on class/instance ids. Specifically, a candidate is a set of points with the same class/instance id. Partial IoU values between each GT candidates and RS candidates are calculated and store in the corresponding cells of a mxn matrix (m: the number of GT candidates, n: the number of RS candidates). Intersection was calculated by counting the number of overlapped point's positions between GT and RS candidates.

Over-segmented case (a GT candidate overlaps with more than one RS candidates), and under-segmented case (more than one GT candidates overlap with the same RS candidate) propose a difficult in assigning optimal IoU components to maximise the overall IoU of the approach. This difficulty can be seen as the assignment problem and was resolved by applying the Hungarian method, which is a classical method for solving the 2-assignment problem in polynomial time. The algorithm was implemented using the "linear_sum_assignment" method from Scipy library with the boolean maximize parameter specified to True for calculating a maximum weight.

The runtime of the key algorithm in each approach was measured using the time.time() method. For performance evaluation, each approach was tested on 5 different input files, with the program run 30 times per files, and the average runtime over 30 runs was reported for each file.

3 Results

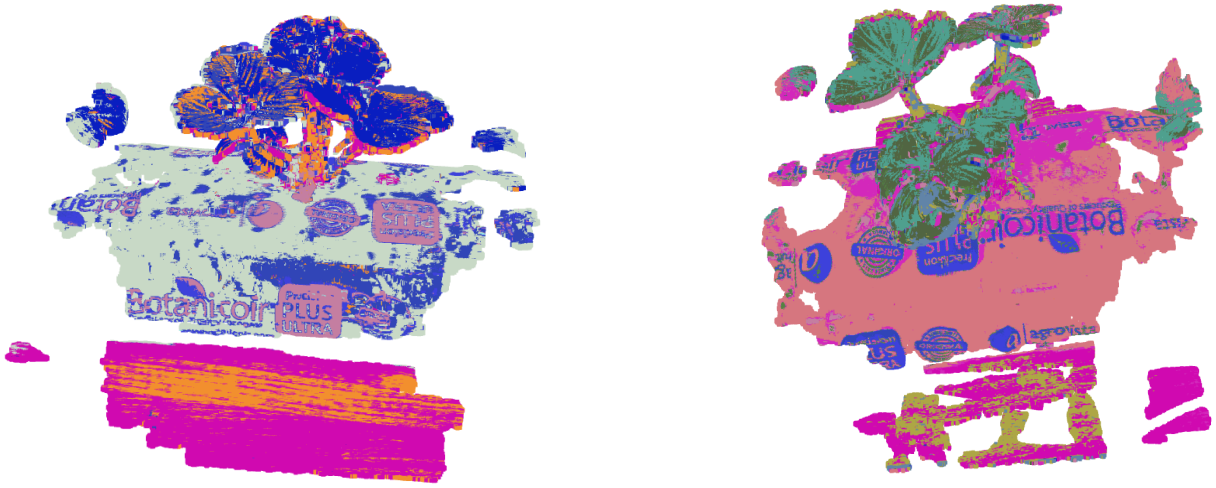
The programs were tested on 5 different test input files: A2_20220512_a.xyz, A2_20220519_a.xyz, A2_20220525_a.xyz, A2_20220531_a.xyz, and A2_20220608_a.xyz. These contain 799854, 963653, 963304, 1013391, and 1292050 points respectively.

I have encountered difficulties when running the generalised DBSCAN (space-and-color combined DBSCAN) on data with more than 5000 points due to quadratic time complexity and memory overflow in the neighbor query step, in which pairwise distance calculation between each point and the others is required. We attempted to parallelise it utilising multithreading and multiprocessing libraries in Python programming language. However, both approaches have not given any desirable speed up compare to the serial version. This outcome is due to the utilisation of the CPython interpreter in the multithreading library, in which a global lock mechanism is employed to allow only one thread to access shared data structure at any given moment, and a demand for an inter-process memory sharing implementation requiring significantly higher resources because information about neighbors needed by cluster assignment is computed in a separate process (Mochurad et al., 2023). For future work, I could consider parallelising it with OpenMP or implementing a GPU-accelerated version using CUDA.

3.1 Accuracy

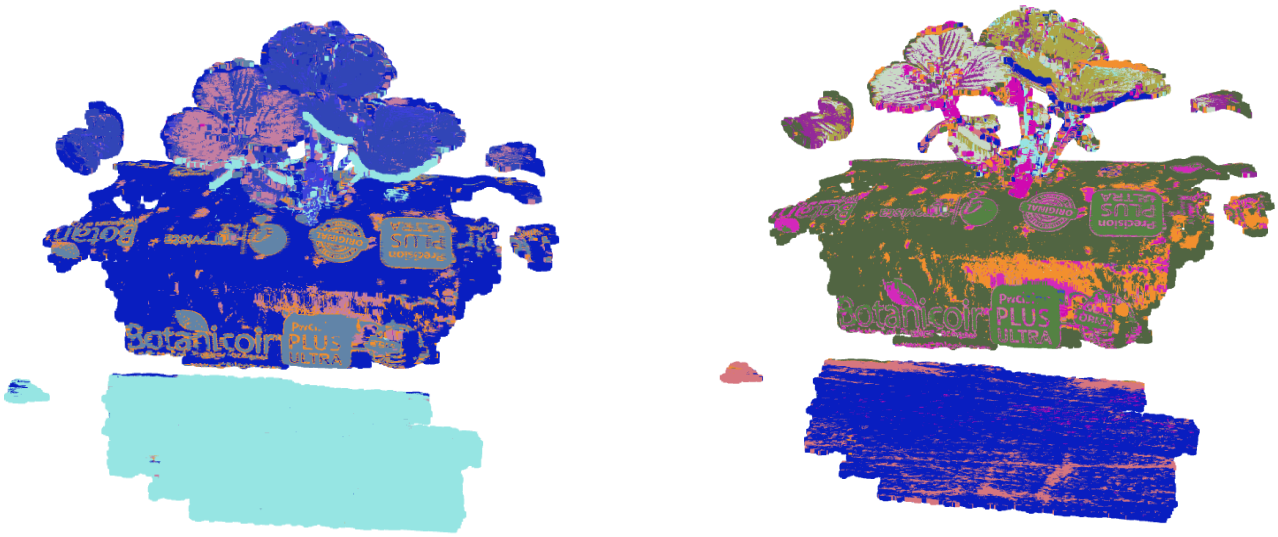
To evaluate the performance of these approaches, both visual representations of segmentation results and indicators calculated from evaluation metrics are utilised. Intersection over Union values of three approaches in both semantic and instance segmentation are specifically shown in tables and visualised in plots for a better comparision.

The following images visualised the segmentation works of each approaches on the A2_20220512_a.xyz input file.



(a) Semantic segmentation (k = 9). (b) Instance segmentation (k = 17).

Figure 4: RGB K-Means Results Visualisation on A2_20220512_a.xyz.



(a) Semantic segmentation (k = 9). (b) Instance segmentation (k = 17).

Figure 5: CIELAB K-Means Results Visualisation on A2_20220512_a.xyz.

Figure 6: DBSCAN Result Visualisation on A2_20220512_a.xyz.

The following tables specifically shows two segmentation types IoU values of the approaches with the five input files.

Table 2: IoU Values for Different Approaches Across 5 Test Input Files (Semantic Segmentation)

File Name	RGB K-Means	CIELAB K-Means	Original DBSCAN
A2_20220512_a.xyz	0.290	0.338	0.291
A2_20220519_a.xyz	0.287	0.315	0.334
A2_20220525_a.xyz	0.316	0.319	0.359
A2_20220531_a.xyz	0.228	0.232	0.394
A2_20220608_a.xyz	0.159	0.215	0.227

Table 3: IoU Values for Different Approaches Across 5 Test Input Files (Instance Segmentation)

File Name	RGB K-Means	CIELAB K-Means	Original DBSCAN
A2_20220512_a.xyz	0.104	0.150	0.202
A2_20220519_a.xyz	0.108	0.131	0.247
A2_20220525_a.xyz	0.079	0.078	0.206
A2_20220531_a.xyz	0.072	0.065	0.213
A2_20220608_a.xyz	0.054	0.057	0.201

The following plots visualised IoU values (used as the performance indicator) of each approaches with the five input files of the approaches (RGB K-Means, CIELAB K-Means, and original DBSCAN).

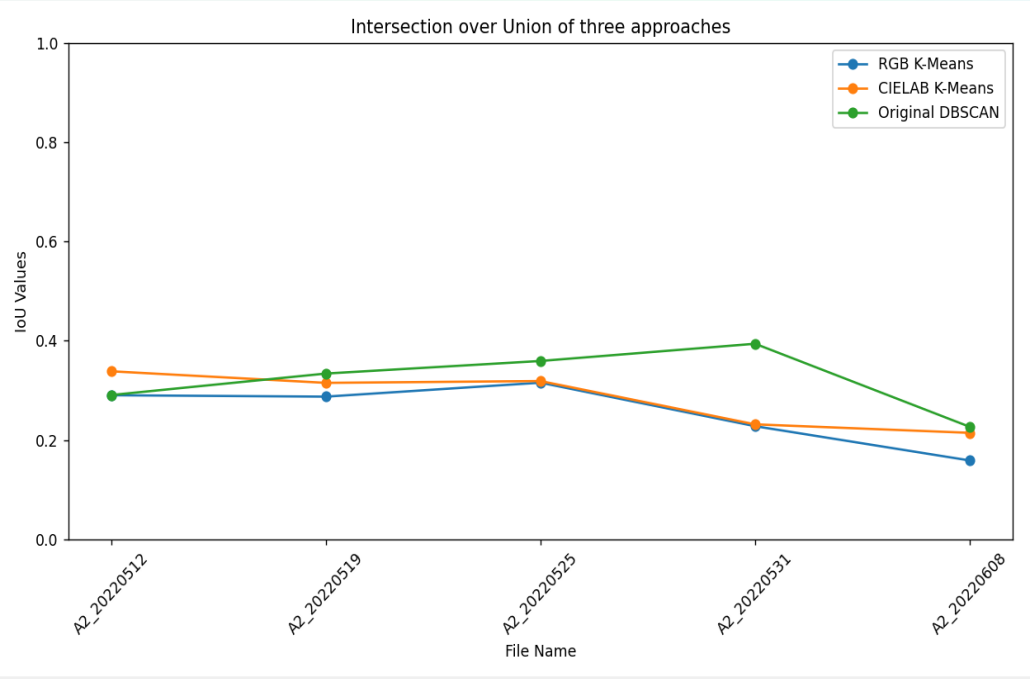


Figure 7: Semantic segmentation IoU values of RGB K-Means, CIELAB K-MEANS, and original DBSCAN with 5 input files.

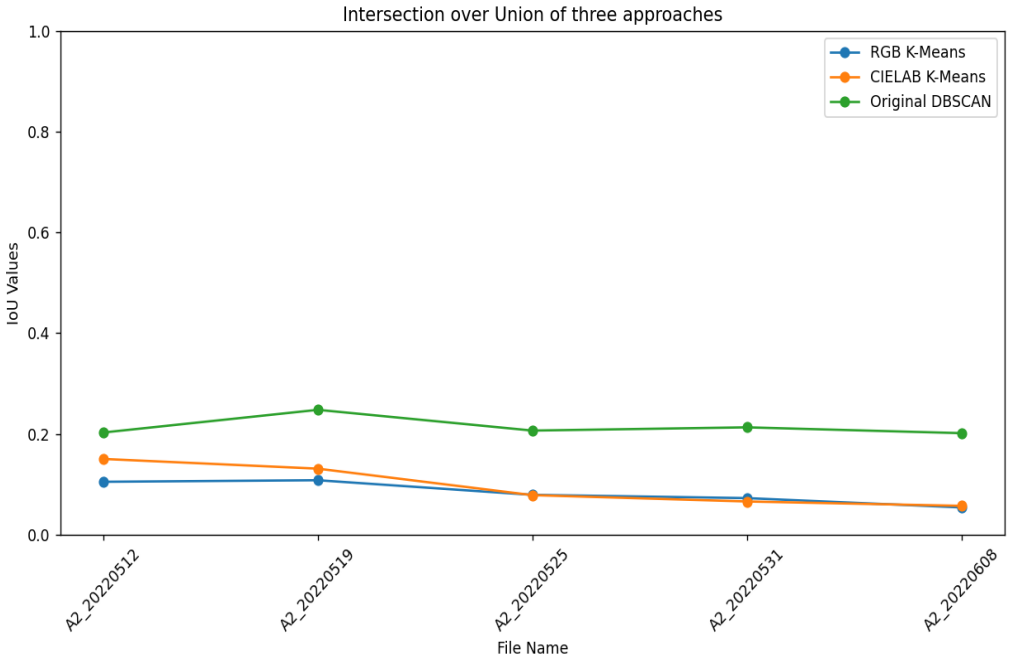


Figure 8: Instance segmentation IoU values of RGB K-Means, CIELAB K-MEANS, and original DBSCAN with 5 input files.

3.2 Runtime

Runtime measurements were also specifically shown in a table and visualised in a plot for comparison.

Table 4: Runtime of Different Approaches Across Input Files (in second)

File Name	RGB K-Means	CIELAB K-Means	Original DBSCAN
A2_20220512_a.xyz	12.7	6.3	64.7
A2_20220519_a.xyz	14.3	4.0	94.4
A2_20220525_a.xyz	16.0	7.5	124.8
A2_20220531_a.xyz	7.5	4.8	119.8
A2_20220608_a.xyz	19.8	7.9	121.6

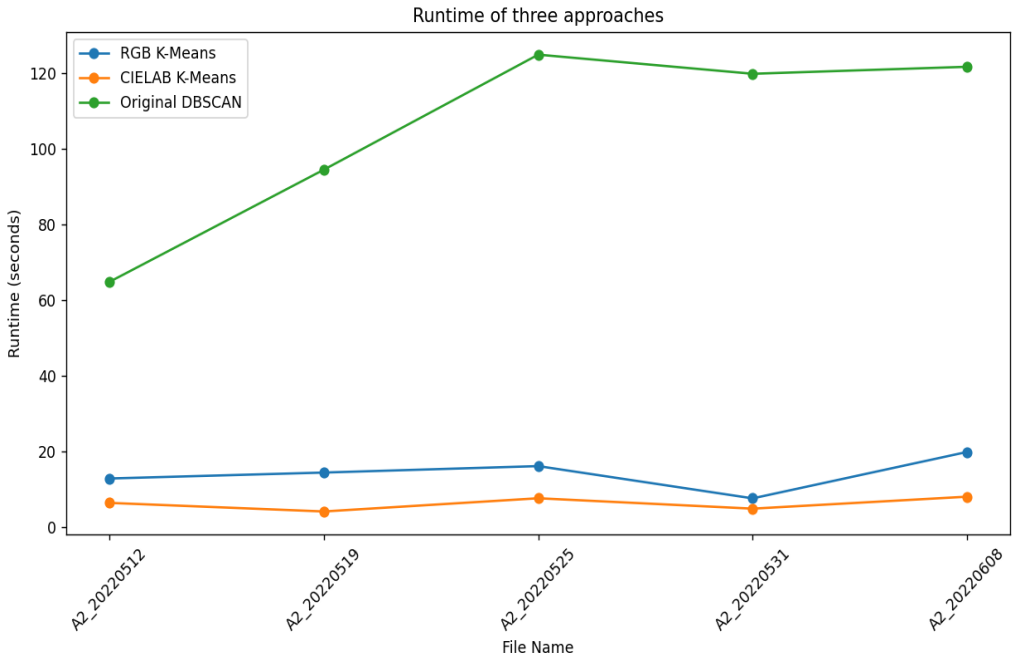


Figure 9: Runtime of RGB K-Means, CIELAB K-MEANS, and original DBSCAN with 5 input files.

4 Discussions and conclusions

The comparison demonstrates that while some algorithms perform better in terms of accuracy, others offer a more efficient solution. Both color-based and density-based approaches yield IoU values within the range of 0.2 to 0.4. However, the spatial density-based method using DBSCAN has the highest accuracy. Among the two K-Means implementation, the use of CIELAB color space provided a slightly more accurate result compared to RGB, highlighting the its closer alignment with human vision and the potential to apply such color spaces in color-based segmentation tasks for a more accurate outcome.

As a trade off, this improved accuracy with DBSCAN comes with a cost of significantly higher computational time compared to the K-Means approaches. This downside makes DBSCAN less practical for large datasets or time-sensitive applications.

The accuracy of DBSCAN can potentially be improved by combining both spatial and color information in forming clusters, which I attempted to implement by modifying the distance metric of the built-in DBSCAN method in Sci-kit learn and manually re-implement DBSCAN's steps with the mentioned distance measurement. However, both of the approaches failed to perform on the large LAST-Straw datasets due to the expensive cost of calculating pairwise distance between points in the set. This problem can be solved by speeding up the region query process. In this project, Grid-based DBSCAN and parallelised DBSCAN have been considered and tested to solve this problem, but still have not gave any results. Grid-based DBSCAN encounters with some limitations as the data used in the project is point cloud data, which requires a multi-dimensional space to represent, especially when both spatial and color information contribute to the distance calculation. Parallelising the region query process attempted using Python did not give any result as mentioned above. It is because the multithreading library in Python prevents the achievement of true parallelism, and the multiprocessing library requires a higher resources for inter-process memory sharing for neighbors information in separate process. The speedup of this process is potentially achieved using OpenMP for a parallelised version, or using CUDA for a GPU-accelerated version in future works.

5 References:

James, K.M.F, Heiwolt, K., Sargent D.J., and Cielniak, G. (2022) Lincoln's Annotated Spatio-Temporal Strawberry Dataset. Available from: https://lcas.github.io/LAST-Straw/. [accessed 3rd July 2024].

Mochurad, L., Sydor, A., and Ratinskiy, O. (2023) A fast parallelized DBSCAN algorithm based on OpenMP for detection of criminals on streaming services, Frontiers in Big Data, 6. Available from: https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2023.1292923/full [accessed 7th September 2024].