



PROJECT REPORT

REAL MASS ESTIMATION USING DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE (DBSCAN) AND K-NEAREST NEIGHBOUR CLASSIFICATION

BONITA NUGROHO WIDJANARKO
18.k1.0001

Faculty of Computer Science
Soegijapranata Catholic University
2022

HALAMAN PENGESAHAN



Judul Tugas Akhir: : REAL MASS ESTIMATION USING DENSITY-BASED SPATIAL
CLUSTERING OF APPLICATIONS WITH NOISE (DBSCAN) AND
K-NEAREST NEIGHBOUR CLASSIFICATION

Diajukan oleh : Bonita Nugroho Widjanarko

NIM : 18.K1.0001

Tanggal disetujui : 06 Januari 2022

Telah setuju oleh

Pembimbing : Hironimus Leong S.Kom., M.Kom.

Penguji 1 : Hironimus Leong S.Kom., M.Kom.

Penguji 2 : Rosita Herawati S.T., M.I.T.

Penguji 3 : Y.b. Dwi Setianto S.T., M.Cs.

Penguji 4 : Yulianto Tejo Putranto S.T., M.T.

Penguji 5 : Yonathan Purbo Santosa S.Kom., M.Sc

Penguji 6 : R. Setiawan Aji Nugroho S.T., MCompIT., Ph.D

Ketua Program Studi : Rosita Herawati S.T., M.I.T.

Dekan : Dr. Bernardinus Harnadi S.T., M.T.

Halaman ini merupakan halaman yang sah dan dapat diverifikasi melalui alamat di bawah ini.

sintak.unika.ac.id/skripsi/verifikasi/?id=18.K1.0001

DECLARATION OF AUTHORSHIP

I, the undersigned:

Name : BONITA NUGROHO WIDJANARKO

ID : 18.K1.0001

declare that this work, titled "REAL MASS ESTIMATION USING DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE (DBSCAN) AND K-NEAREST NEIGHBOUR CLASSIFICATION", and the work presented in it is my own. I confirm that:

- 1 This work was done wholly or mainly while in candidature for a research degree at Soegijapranata Catholic University
- 2 Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- 3 Where I have consulted the published work of others, this is always clearly attributed.
- 4 Where I have quoted from the work of others, the source is always given.
- 5 Except for such quotations, this work is entirely my own work.
- 6 I have acknowledged all main sources of help.
- 7 Where the work is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Semarang, January, 10, 2022



BONITA NUGROHO WIDJANARKO

18.K1.0001

STATEMENT PAGE OF SCIENTIFIC PUBLICATIONS FOR ACADEMIC INTEREST

I, the undersigned:

Name : BONITA NUGROHO WIDJANARKO
Study Program : Informatics Engineering
Faculty : Computer Science
Type of Work : Thesis

approve to grant Soegijapranata Catholic University Semarang Non-exclusive Royalty-Free Rights for the scientific works entitled "REAL MASS ESTIMATION USING DENSITY-BASED SPATIAL CLUSTERING OF APPLICATIONS WITH NOISE (DBSCAN) AND K-NEAREST NEIGHBOUR CLASSIFICATION" along with existing tools (if needed). With this Non-exclusive Royalty-Free Rights, Soegijapranata Catholic University has the right to keep, transfer media/format, manage in the form of a database (database), maintain, and publish this final project as long as it includes my name as a writer/creator and as a Copyright owner.

Semarang, January, 10, 2022



BONITA NUGROHO WIDJANARKO

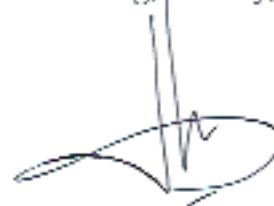
18.K1.0001

ACKNOWLEDGMENT

My first yet big appreciation, I would like to express my sincere gratitude to my supervisors Hironimus Leong, S.Kom., M.Kom. for constant support and help during the writing of this thesis. Without his great feedback and excellent guidance this thesis would not have been completed.

Last but not least, I would give my warm but heartfelt thanks to my friends and family for showering their tremendous love and hope with excellent valuable suggestions whenever I need them. Thank You all for the strength you gave me.

Semarang, January, 10, 2022



BONITA NUGROHO WIDJANARKO

18.K1.0001

ABSTRACT (Style : Abstract Title)

Many real-world applications can be acquired from efficient and effective mass measurement. However, further development is needed for our current mass measurement. Estimating an object's mass through the naked eye is not possible and it's very inconvenient to bring a measurement scale everywhere just to calculate the mass of an object. The proposed study shows that there's no need for human intervention to calculate an object's mass by placing the object one by one into the mass scales, it can be done just by taking pictures of them.

Using the object's top view and side view photos, the object's mass can be found. Image processing is needed to achieve a clean edge of the object and replace the object's background with transparency. There are two sections in this model which are object identification and object volume measurement. In the object identification model, processed side view image will be combined with the train data and clustered by the DBSCAN algorithm. A selected cluster of the input image will be used as k-NN classification train data. k-NN will find the nearest neighbor of the input image among the input image's cluster. The object is identified and the object's density is obtained.

In the volume estimation model, an area calculation and height estimation are required. Object's area is obtained by calculating the non-zero pixels of the thresholded object's top view. While object's height is estimated by drawing a bounding box around the object of the object's side view image. After the object's area and height are obtained, multiplying them will get the object's volume. Object's density and an object's volume have been acquired and by multiplying them, the object's mass is calculated.

It was found that this model is able to estimate an object's mass with acceptable error due to small datasets and manual measurements which increase the possibility of human error. The average accuracy of multiple applications reaches 90,2% and this model works best with light color objects. The highest accuracy is obtained with a 19 cm distance from the camera to the object.

Keyword: mass measurement, image processing, object identification, object's density, volume estimation.

TABLE OF CONTENTS

APPROVAL AND RATIFICATION PAGE (Heading plain).....	ii
DECLARATION OF AUTHORSHIP.....	iii
ACKNOWLEDGMENT.....	iv
ABSTRACT (Style : Abstract Title).....	vi
LIST OF FIGURE.....	ix
LIST OF TABLE.....	x
CHAPTER 1 INTRODUCTION.....	1
1.1. Background.....	1
1.2. Problem Formulation.....	2
1.3. Scope.....	2
1.4. Objective.....	2
CHAPTER 2 LITERATURE STUDY.....	3
CHAPTER 3 RESEARCH METHODOLOGY.....	11
3.1. Data Collection.....	11
3.2. Algorithm.....	11
3.3. Design.....	12
3.4. Coding.....	13
3.5. Analysis.....	14
CHAPTER 4 ANALYSIS AND DESIGN.....	15
4.1. Dataset Preparation.....	15
4.2. Dataset Preprocessing.....	18
4.2.1. Input Image Preprocessing.....	18
4.2.2. Dataset Preprocessing.....	19
4.3. DBSCAN.....	21
4.4. K-NN.....	22
4.5. Object's Volume Estimation.....	23
4.6. Object's Mass Estimation.....	24

4.7. Analysis.....	25
CHAPTER 5 IMPLEMENTATION AND RESULTS.....	27
5.1. Implementation.....	27
5.1.1. Remove Input Image Background to Transparent.....	27
5.1.2. Extract The Zipped Dataset.....	29
5.1.3. Image Hash.....	30
5.1.4. Distance Matrix.....	31
5.1.5. Coordinates Form.....	31
5.1.6. DBSCAN Algorithm.....	31
5.1.7. Input Image Cluster.....	36
5.1.8. k-NN Algorithm.....	37
5.1.9. Get Object's Density.....	38
5.1.10. Volume Calculation.....	39
5.1.11. Object's Mass Estimation.....	41
5.2. Results.....	41
5.2.1. DBSCAN Algorithm.....	41
5.2.2. Object Identification.....	42
5.2.3. Object's Area.....	43
CHAPTER 6 CONCLUSION.....	50
REFERENCES.....	51
APPENDIX.....	a

LIST OF FIGURE

Figure 4.1: Data Preprocessing Result.....	19
Figure 4.2: Mask Transformation.....	19
Figure 4.3: Real Mass Estimation Flowchart.....	26
Figure 5.1: Cluster and minPts Regression Linear 1.....	47
Figure 5.2: Cluster and minPts Regression Linear 2.....	47
Figure 5.3: Cluster and minPts Regression Linear 3.....	48
Figure 5.4: Cluster and minPts Regression Linear 4.....	48

LIST OF TABLE

Table 4.1 : Fruits and Vegetables Kinds Dataset Table.....	15
Table 4.2 : Kaggle Dataset Examples.....	16
Table 4.3 : Input Image Examples.....	17
Table 5.1 : DBSCAN Time and Cluster Test Result 1.....	41
Table 5.2 : DBSCAN Time and Cluster Test Result 2.....	42
Table 5.3 : Object Identification Result of 10 Fruit Kinds.....	42
Table 5.4 : Object Identification Result of 20 Fruit Kinds.....	42
Table 5.5 : Mass Estimation Result of 6 Apples.....	43
Table 5.6 : Mass Estimation Result of 6 Lemons.....	44
Table 5.7 : Mass Estimation Result of 6 Oranges.....	45
Table 5.8 : Average Accuracy of Apples	46
Table 5.9 : Average Accuracy of Lemons	47
Table 5.10 : Average Accuracy of Oranges.....	47

CHAPTER 1

INTRODUCTION

1.1. Background

Mass measurement is one of the major necessities in daily life and has many real-world applications. A lot of benefits can be acquired by knowing the object's mass but further developments are needed for our current mass measurement. The mass measurement itself has a linear relationship on its own scale where the bigger mass to measure requires a bigger scale too. Estimating an object's mass through naked eye is not possible and it's very inconvenient especially when a person needs to bring a measurement scale everywhere just to calculate the mass of an object.

Practicality and efficiency have become our primary necessities on this very era. A new mass estimation approach is needed and with the help of modern technologies, it can be realized. Computer vision and machine learning have been incredibly advanced during these past few years. By featuring them there's no need for human intervention to place the object one by one into the mass scales, it can be done just by taking pictures of them.

Most of the research that has been conducted still needs further improvement and a lot of high resources. High-tech cameras and 3D images have become essential. Nowadays people tend to bring their own personal devices such as mobile phones everywhere. By using a mobile phone's built-in camera users can utilize it to take the object's photo and use them for mass measurement. Moreover, it can lead us to bigger things such as the calculation of food calories from their mass, mass count for delivered orders, cargo purposes, determining what tools we should use to carry those objects and many more.

We all need a simple device with great performance but low resources. From this research just by phone camera, users can estimate the object volume and mass. This research's main purpose is with new methodology just by using 2D images and phone camera, we can know the object's volume and mass, there is no need for special devices nor high-tech cameras.

In this research the author has focused on fruits and vegetables as our object's mass measurement. Reasons behind are because fruits and vegetables have important roles in human lifestyle, besides for weighing fruits and vegetables manually can be very laborious, there has to be an easy way for humans to calculate the mass estimation. New mass

measurement can be done just by taking the top view image and side view image using the mobile device's built-in camera of the object in order to find the volume. With the help of machine learning and mathematical formulas, an object's mass can be found.

Clustering and classification take part in this research. In order to have an accurate object detection, clustering is done in order to become training data for classification. New algorithm approach that features clustering by DBSCAN and classification by k-NN algorithm is expected to be more cost effective. As for the evaluation method the author uses accuracy measures which consist of comparison between real object's mass and obtained mass. Accuracy measures value become the benchmark of this research success.

1.2. Problem Formulation

The following questions is accordance with the focus of this research.

1. How to use and how qualified is the DBSCAN algorithm for clustering?
2. Does various image distances to object affect the result?
3. How accurate is the result compared to real object's mass?

1.3. Scope

Dataset in this research is obtained through Kaggle which consists of fruits and vegetables catalog images. This dataset consists of 131 kinds of fruits and vegetables, each with approximately 400-900 training images. Kaggle dataset becomes training data for clustering by DBSCAN while the results become training data for classification by k-NN in order to accelerate object detection.

For evaluation methods, accuracy measures will be applied. Through the accuracy measures the author can find out how good the model can perform, how well it is compared to other algorithms, and measure how accurate the results are.

1.4. Objective

This research purpose is to find a new mass measurement method which is more effective and efficient than the previous one by using DBSCAN and k-NN approach. With the new measurement method, no laborious human intervention is needed to measure the fruit one by one. Mass measurement can be done just by taking pictures of the object using a built-in mobile phone camera where in this research fruits and vegetables as the target object.

CHAPTER 2

LITERATURE STUDY

Standley et al. in [1] focus on proposing a model for estimating one such physical property, weight, from an object's image. The authors used the datasets from Amazon.com's products information that contain about 3 million products, each with its weight, dimensions, an image and also household datasets that's manually weighed on an appropriate scale and each of their measured dimensions. Extensive filtering is still being needed for The large Amazon test set and for the household test set were segmented using the GrabCut algorithm and placed on pure white backgrounds in order to achieve a high-quality dataset. Two neural network tower were developed which are density tower and volume tower beside there is geometry module to complete object's volume understanding. Weight prediction can be found by combining both towers. On the other side, for experiment algorithms that they use for baseline models are k-Nearest Neighbors and pure CNN Baseline. The result of their experiments is outstanding, which can be said to be similar compared to humans, while poor K-NN performance proved that semantically similar objects often have massive different weights.

This experimental results on the large-scale Amazon dataset as well as the household dataset validate their modular approach, showing a significant improvement over CNN baselines and performance that is competitive with humans. Trevor et all said that the limitation of their research is images alone are not sufficient, such as an image of a toy car could be indistinguishable from an image of a real car, so it's concluded that more dataset is needed. The differences with this research are on the dataset and algorithm.

Kollis et al. research in [2] talks about pig's weight estimation using image analysis and statistical modeling. A full set of data containing 84 image weight pairs is required for this research. The base dataset is formed by 38 unique animals that were identified. The efficiency of the weighing process can be improved with an automated weighing system which is less time-consuming and laborious. The algorithms that take part in this research are object detection, segmentation, filtering, and feature extraction.

Area and neck-to-tail length had a high correlation with a pig's weight based on the investigated feature. A weaker relationship has been found between the spine length feature and weight. A lot of improvement is needed due to time constraints and technical problems. Real-time automated system to capture pigs images and weights estimation was not developed.

Still, due to the success of the individual components of this project, it laid strong foundations for future work and such a system would be relatively easy to produce. Due to the limitation of a small sample of pigs and small weight range this study can not perform optimally.

Konovalov et al. [3] review about automatic weight estimation of harvested fish from images. There are three datasets being used which are The Barra-Ruler-445 (BR445) dataset which contains 445 images with manually measured weights with a range of 1 – 2.5kg. The second dataset was Barra-Area-600 (BA600). This dataset contains more than 600 image-weight pairs, where BA600 fish weights were between 0.2 kg and 1 kg. The third one is a dataset (denoted BW1400) containing 1,400 harvested barramundi images all with weight values from the 0.15 – 1.0 kg range. To minimize dependency on such transient colors, in training and testing, all images were transformed to grayscale. This research was approached by weight-from-area and weight-from-image models. The algorithm used is LinkNet-34 as a more advanced segmentation Convolutional Neural Network (CNN). The author successfully tested a conversion of a segmentation CNN, LinkNet-34, to weight-predicting CNN, LinkNet-34R, achieving 4-11% test MAPE values. Remarkably, the two simple mathematical models based on the whole-fish silhouette generalized better when applied to the unseen test images from the different geographical locations. The second main question was answered by demonstrating that the simplest one-factor mathematical model performed better than the two-factor model on the new test images. The limitation is that only the no-fins version of the direct regression via LinkNet-34R performed well on the test images, this result strongly indicating the possibility of overfitting of the whole-fish version.

Nystad [4] studies automated detection and weight estimation of fish in underwater 3D images. Video made up of 809 images of salmon swimming inside of a fish tank become their main dataset. Since fish image is captured inside the pond, by using the mathematical formula, distance can be found. An image dataset of fish in a tank is featured, and this model performs in finding a robust segmentation method for detection of fish and further this model is used as a basis for weight estimation. Active Shape Model (ASM) implementation with MATLAB is built as their main tool. ASM has the main job as a backend to handle the problem of segmenting salmon in the given images.

An image segmentation method is proposed which is based on active shape models to extract the contour of fish in images from an underwater range-gated 3D camera. This study suffers from not knowing the correct camera parameters to project from image space to world space, besides the fish's real size in the dataset are unknown.

Besides, in case the threshold set on value for the Mahalanobis metric is too small, only a minority of the fishes can be segmented. But if the threshold set value is too high, the contour of the fishes can not be fully found and the effect is a lot of false positives. ASM is great for extracting the length and height of an object based on known landmark positions, unfortunately ASM is an edge based segmentation algorithm, which means that they perform on gradients in the image making it loses information about the intensity inside of the object's contour. Finding a robust segmentation algorithm for detecting fish in images and handling noisy depth information is complicated, which is to project the size of the fish from image space to world space. Small training data affect the result either.

In this research Sanchez, German et al. [5] discuss automatic measurement of fish weight and size by processing underwater hatchery images. The length and weight of 150 fish specimens of *Oreochromis niloticus* belonging to the Cichlidae family were registered in this manner, and used for testing the proposed methodology. They use a combination of homomorphic filtering, contrast limited adaptive histogram equalization (CLAHE) and guided filtering for fish image enhancement since image object border is crucial. Using a combination of 2D saliency detection and morphological operators the fish images were segmented. Finally, the fish length was obtained by using a third-degree polynomial regression on the fish mid-points. The length was calculated to estimate the weight with several regression algorithms. This model was proved to be the most appropriate method for fish weight based on length regression.

Combining homomorphic filtering for lighting balance, CLAHE for contrast enhancement and guided filtering for noise reduction and border highlighting. The use of CLAHE in combined HSV-RGB color space gave particularly good results when executed on the dataset. The proposed algorithm showed remarkable effectiveness when carrying out fish segmentation. The limitations they had were problems such as lighting inequalities, poor contrast and noise are obstacles to the implementation of computer vision technologies. In this case, the requirement of segmenting fishes before measurement made it necessary to work with images where these issues were managed, which is hard to accomplish in uncontrolled settings but results showed that relatively simple methods, such as polynomial regression, can be employed for calculating the fish length and estimating its weight with remarkable accuracy.

Lines et al. in [6] has shown an automatic image-based system for estimating the mass of free-swimming fish. Stereo pair video images are collected using a vertically oriented

pair of video cameras taken as the dataset with a baseline around 0.5 m with optical axes converging 1.5 m in front of the cameras. Images of salmon are collected between 1 and 2 m from the cameras. The authors use the Binary Pattern Classifier and Point Distribution Model (PDM) algorithm.

This research gives a great idea that even though the area is not clear, using the PDM algorithm can be helpful considering there's no control over natural environments as the background. The help of black blackboard can help provide a more uniform background. Salmon in a tank is being tested by these components and have been shown to produce estimates of fish mass, the results are well-correlated with the measured mass. Using a combination of generic and ad-hoc techniques, further improvements in the accuracy, robustness, and error detection will be sought. This development and testing is dependent on the collection of the expanded database. These components images contain a usable image of a salmon to be identified, enable the self-occluding boundary of the salmon to be identified in 3D space and enable the mass of the salmon to be estimated from information about this boundary. More robust fitting of the model is needed. An image fitting is very important since it's become the base of the research and affects the result accuracy either.

This research by Hamdan et al. in [7] stated about sugar cane mass estimation from images using deep neural networks and sparse ground truth. The camera has a fixed sampling frequency that is fast enough to measure all the material passing by, it is mounted at a fixed distance from the elevator, and the only additional factor needed is slat velocity to scale the accumulated mass. Intuitively, the velocity is scaling the mass to produce a mass flow, since if the mass was sitting still it would not be accumulated. Even though complex nonlinearities exist with the density and the material location in the image (e.g. the same material further away is smaller in the image), these patterns can be learned by optimizing a DNN due to fixed locations between the camera and the elevator.

The authors use DNN using only final (aggregated) load weights as their base algorithm. A 9-layer DNN with ELU units and residual connections, defined as – “RES9-ER”, was found to have good predictive accuracy as well as fast training and inference. A semi-supervised algorithm that makes inference on the mass flow of material from images by training a DNN with sparse ground truth, and showed improvements over older and more expensive methods that acquire a volumetric estimate of the material and calibrate the density. No extreme outliers were found in the validation or test sets, but the training set contained a single outlying run circled in red. The total mass prediction of the outlying run was too low

(40% of what it should have been), but yet the DNN mass prediction per image before scaling with elevator speeds and capture time was very high. It is likely that the optimization process tried to use the reflections of bamboo on the side of the elevator to compensate for the low total predicted mass. Since this outlier was an extreme corner condition of very low elevator speed that is not at all likely in real applications, thus it could be removed and the network retrained to likely achieve better overall results.

Bhatt et al. [8] has shown barqi breed sheep weight estimation based on a neural network with regression. The authors use the images of 52 sheep from a high-definition mobile camera. They put their focus to minimize human involvement in measuring the weight of the sheep and making the difficult task of estimating sheep weights quick and accurate. They are using a novel approach for segmentation and a neural network-based regression model for achieving better results for the task of estimating sheep's weight. They use Segmentation and Neural Network Based Regression Model (ANN (Artificial Neural Nets) for regression tasks). The proposed weight estimation system consists of three phases which are preprocessing phase, segmentation phase, weight estimation.

They take into consideration the age and gender of the sheep which their system takes care of and enables them to get better accuracy than the other similar approaches. Unfortunately, the task of segmentation seems to be difficult here since the haystacks in the background sometimes merge with the texture of sheep wool. However, this problem has been successfully solved in a newer segmentation ANN helped us in modeling a pretty decent approximation of the underlying function. Using this technique we will be able to estimate the sheep's weight with a simple image from a mobile device. This technique is different from the previous approaches as the previous approaches were either dependent on advanced system setup or high-tech cameras.

Chaithanya C. and Priya S. [9] write an article about object weight estimation from 2D images. It has two phases, a training phase, and a classification phase. Color, texture, shape, and size features are given as the input for the classification phase. Via a special calibration technique, using the built-in camera of such mobile devices the user can record photos of the objects. The proposed system uses 2D images to estimate the weight of the object in the image. They use Image Processing and Object Recognition. This research is based on the K-Means algorithm for clustering and the Mahalanobis Distance classifier (MMDC) algorithm for classification. MMDC requires fewer training samples and does not suffer from overfitting problems.

In order to reduce the large latency and meet real-time requirements, a novel distributed Canny edge detection algorithm is used, which has the ability to compute the edges of multiple blocks at the same time. To support this, an adaptive threshold selection method is used. The performance of the proposed system can be improved by using a better shape recognition method so that volume can be calculated from a single image using the appropriate volume calculating formula, which can result in accurate volume and weight calculation. The proposed method of weight calculation gives approximately 97 percent accuracy. From the results, it is clear that nearly an equivalent weight is obtained for each object.

Jiang and Guo [10] brought up the bodyweight analysis from human body images. A visual-body-to-BMI dataset is collected and cleaned to facilitate the study, which contains 5900 images of 2950 subjects along with labels corresponding to gender, height, and weight. Among this information, body weight is a good indicator of health conditions. For this topic, they use Deep networks for contour and skeleton joints detection, convolutional pose machines (CPM), Support vector machines (SVMs), Gaussian processing regression.

To address the visual bodyweight analysis problem, the computational method of five anthropometric features is developed. The errors of the three estimation tasks evaluated by several measurements are within acceptable ranges. Comparing with the facial images analysis approaches, the proposed method performs better in most cases. Furthermore, our anthropometric features significantly outperform the VGG-Net feature on BMI estimation. Unfortunately, The performance of the overweight category is a little lower than the obese. The prediction accuracy of the underweight category is the lowest since there are only 46 body images in the database that belong to the underweight category, among them, 9 are in the test set and 37 are in the training set. The lack of enough underweight body images in the training set could be the reason for this lower performance. Nevertheless, based on all experimental results, it is promising to analyze body weight or BMI from the 2D body images visually.

Clustering algorithms are attractive for the task of class identification in spatial databases. However, the well-known algorithms suffer from severe drawbacks when applied to large spatial databases. This paper Ester et al. [11] is talking about the new clustering algorithm Density-Based Spatial Clustering of Applications with Noise (DBSCAN) which relies on a density-based notion of clusters, this algorithm is designed to discover clusters of arbitrary shape. DBSCAN requires only one input parameter and supports the user in

determining an appropriate value for it. They perform an experimental evaluation of the effectiveness and efficiency of DBSCAN using synthetic data and real data of the SEQUOIA 2000 benchmark. Their research focuses on introducing and implementing the DBSCAN algorithm thus comparing DBSCAN with the CLARANS algorithm. This article is useful for my research as I know what DBSCAN clustering is and what the advantages and disadvantages are, how to implement DBSCAN, and knowing deeper about the DBSCAN algorithm.

DBSCAN is significantly more effective in discovering clusters of arbitrary shape than the well-known algorithm CLARANS, and that DBSCAN outperforms CLARANS by a factor of more than 100 in terms of efficiency. First, they have only considered point objects. Spatial databases, however, may also contain extended objects such as polygons. We have to develop a definition of the density in an Eps-neighborhood in polygon databases for generalizing DBSCAN. Second, applications of DBSCAN to high dimensional feature spaces should be investigated. In particular, the shape of the k-dist graph in such applications has to be explored. Overall DBSCAN has an outstanding result. The results of these experiments demonstrate that DBSCAN is significantly more effective in discovering clusters of arbitrary shape than the well-known algorithm CLARANS. Furthermore, the experiments have shown that DBSCAN outperforms CLARANS by a factor of at least 100 in terms of efficiency.

Huge differences among the other researchers, most of the research uses animals as their object mass measurement, their mass measurement aims are pigs in [2] and fish [3]-[6]. Besides living objects, there is research that has been conducted measuring the mass of daily objects based on amazon's product information [1]. Standley et al. research requires a massive dataset considering there are a lot of daily objects. They combine the k-NN algorithm with CNN Baseline. While our main focus on this research is to measure the fruits and vegetables mass considering fruits and vegetables have an important role in human life and this research uses DBSCAN and k-NN for the algorithm. While on the second journal by Kollis et al., [2] they tend to do image processing like object detection, segmentation, filtering, and feature extraction for pig's weight estimation. Third journal by Konovalov et al. [3] estimates fish weight by using a more advanced segmentation Convolutional Neural Network (CNN) which is LinkNet-34. Fourth journal by Nystad [4] estimates salmon weight by using Active Shape Model (ASM) algorithm. Sanchez, German et al. [5] chose fish as their object focus using a combination of homomorphic filtering, contrast limited adaptive histogram equalization (CLAHE) and guided filtering for fish image enhancement.

Sixth journal by Lines et al. [6] estimated free-swimming fish mass with the Binary Pattern Classifier and Point Distribution Model (PDM) algorithm. For the seventh journal [7] Hamdan et al. has a different object focus which is sugar cane mass by using deep neural networks and sparse ground truth. Another research Bhatt et al. [8] conducts barqi breed sheep weight estimation based on a neural network with regression. By using a simple image from a mobile device, sheep's weight can be estimated accurately. Similar research has been conducted by Chaithanya C. and Priya S. about weight measurement [9] using K-means and MMDC classifiers. On this research by using a different modern approach with DBSCAN and k-NN it's expected to be less time-consuming and effective. As for the tenth [10] journal Jiang and Guo discuss bodyweight analysis from human body images by using Deep networks for contour and skeleton joints detection, convolutional pose machines (CPM), Support vector machines (SVMs), Gaussian processing regression. Previous research [1]-[10] haven't used the DBSCAN algorithm as their algorithm. DBSCAN is a great clustering algorithm but still has less popularity than another algorithm like K-means. Ester et al. in [11] talk about the DBSCAN algorithm and advantages among the other algorithms like CLARANS.

CHAPTER 3

RESEARCH METHODOLOGY

3.1. Data Collection

This research required fruits and vegetables images dataset for training the algorithm. The dataset that are being used in this research were obtained through Kaggle by title Fruits 360 dataset: A dataset of images containing fruits and vegetables as with 2020.05.18.0. for the version. The dataset set can be obtained simply by downloading the dataset bundle and the 1.31 GB zip package already consists of all the mentioned below. Available fruits and vegetables image dataset cover 360 degree images with original size and scaled size which is 100x100 pixels for the image size all with extracted background.

Training set and test set have already been incorporated once you download the package. This data set comprises 131 kinds of fruit with the total number of images 90483. It is divided into a training set which consists of a total 67692 images and a test set with a total 22688 images. As for the filename format "r" stands for rotated fruit as an example imageindex100.jpg (e.g. 32100.jpg) or rimageindex100.jpg (e.g. r32100.jpg) or r2imageindex100.jpg or r3imageindex100.jpg. "r2" stands for the fruit was rotated in the 3rd axis and "100" comes from image pixels size which are 100x100 pixels. Dataset images come in RGB values. In this research only the training set is used.

As for the density table, fruits and vegetables density obtained from various sources which has the best accuracy for particular fruits and vegetables. Density table formed in csv file format. 2D input images in this research are obtained by using the iPhone X built-in camera as our device.

3.2. Algorithm

Clustering and classification takes an important part of this research as to achieve an accurate object detection. The author is going to use both algorithms for clustering in order to make classification training simpler where classification only trained data where the cluster has similar features with the input image. At first, the author sought K-means for clustering algorithm but K-means usage is overrated already and depends on 'k' value while DBSCAN doesn't require a specific number of clusters. Algorithms that featured in this research are DBSCAN and k-NN as DBSCAN for clustering and k-NN for the classification. Although DBSCAN is less popular among the other algorithms, DBSCAN is great at clustering.

Reasons behind the DBSCAN algorithm is a great help in clustering is because noise doesn't affect the algorithm, DBSCAN can find a cluster completely surrounded by another cluster, and DBSCAN can discover an arbitrarily shaped cluster. In this paper, the clustering algorithm of DBSCAN which relies on a density-based notion of clusters is being presented. The DBSCAN algorithm requires only two input parameter and the algorithm supports in determining an appropriate value for it. Another advantage of DBSCAN is that there is no requirement for using training data. DBSCAN doesn't perform prediction, the algorithm only tries to cluster similar data points so it performs clustering on the actual data. DBSCAN also looks into the spatial density of data points which makes DBSCAN different among other clustering algorithms. Additionally, DBSCAN is robust toward outlier detection all based on the data points density matrix, low density areas are separated as the outliers.

After the data has already been clustered it becomes classification training so that image classification can be done easily and fast. The clustered part where the cluster is similar to the related input image is taken for classification training data. As for the classification, the author uses k-NN because a lot of research before that has been conducted using k-NN has great results besides k-NN is a great classification algorithm, there's disadvantages with using k-NN because k-NN doesn't perform well with large dataset which the author covered by clustering the dataset first using DBSCAN and k-NN is famous for lazy learner which there is no training period in as a result fast prediction can be accomplished.

Both of the algorithm applications in order to obtain accurate object detection, after the object has successfully detected the result will be matched into a density table and by using a mathematical formula along with OpenCV for volume estimation, the object's mass can be found.

3.3. Design

Clustering and classification takes an important part of this research as to achieve an accurate object detection. The author is going to use both algorithms for clustering in order to make classification training simpler where classification only trained data where the cluster has similar features with the input image. At first, the author sought K-means for clustering algorithm but K-means usage is overrated already and depends on 'k' value while DBSCAN doesn't require a specific number of clusters. Algorithms that featured in this research are DBSCAN and k-NN as DBSCAN for clustering and k-NN for the classification.

Although DBSCAN is less popular among the other algorithms, DBSCAN is great at clustering. Reasons behind the DBSCAN algorithm is a great help in clustering is because noise doesn't affect the algorithm, DBSCAN can find a cluster completely surrounded by another cluster, and DBSCAN can discover an arbitrarily shaped cluster. In this paper, we presented the clustering algorithm DBSCAN which relies on a density-based notion of clusters. It requires only one input parameter and supports the user in determining an appropriate value for it. Another advantage of DBSCAN is that there is no requirement for using training data. DBSCAN doesn't perform prediction, the algorithm only tries to cluster similar data points so it performs clustering on the actual data. DBSCAN also looks into the spatial density of data points which makes DBSCAN different among other clustering algorithms. Additionally, DBSCAN is robust toward outlier detection all based on the data points density matrix, low density areas are separated as the outliers.

After the data has already been clustered it becomes classification training so that image classification can be done easily and fast. The clustered part where the cluster is similar to the related input image is taken for classification training data. As for the classification, the author uses k-NN because a lot of research before that has been conducted using k-NN has great results besides k-NN is a great classification algorithm, there's disadvantages with using k-NN because k-NN doesn't perform well with large dataset which the author covered by clustering the dataset first using DBSCAN and k-NN is famous for lazy learner which there is no training period in as a result fast prediction can be accomplished.

Both of the algorithm applications in order to obtain accurate object detection, after the object has successfully detected the result will be matched into a density table and by using a mathematical formula along with OpenCV for volume estimation, the object's mass can be found.

3.4. Coding

Python ver 3.8.2 becomes this research main computing language. Major reasons behind the author choosing python are because python is a really compatible machine learning language with various tools provided by python. Python simplicity and consistency has proven by the popularity of this computing language among the AI and machine learning communities. Wide variety of great libraries can be accessed easily to help develop AI and machine learning models.

Besides, python has incredible visualization tools where users can visualize their dataset and by visualization tools it accommodates data analysis better. Besides various features provided by python, python is human-friendly, easy to understand and implement makes python the right choice for machine learning.

As for the density table the author chose to use in csv form in order for easier human readability and relatively easy to manipulate csv file. With the help of pandas csv can be read fast by python.

3.5. Analysis

The author uses 131 kinds of fruits with a total of 67692 images. Further analysis required in order to measure the success of the algorithm for object detection. Compatible dataset is needed to achieve high accuracy. The author will use the analysis of 10, 20, 30, and 40 kinds of fruits and vegetables. How many kinds of fruits and vegetables the algorithm can read accurately. Besides, the DBSCAN algorithm has particular variables such as ϵ and minPts which need to be defined and support the clustering accuracy. On the other hand, input images have major roles. Another analysis is needed to find if different lighting and background affect the image clarity and detection or not which we discussed deeper in the next section. As for evaluation, accuracy measure about comparison between actual mass and estimated mass will be applied.

CHAPTER 4

ANALYSIS AND DESIGN

This study project is proposed to estimate an individual object's real mass by using 2D images with the help of DBSCAN clustering and k-NN classification. By using object detection and image processing, the volume of the object can be calculated and matched with the density table as the final result, the object's mass can be estimated. This goal is approached with the following steps :



















































4.1. Dataset Preparation

There are two sections to this model, the first one is image identification and the second one is volume and mass estimation. To perform this study, the fruits and vegetable dataset from Kaggle take major roles as the first section's main dataset. To identify the success of the first section which is image identification, 131 kinds of fruits and vegetables from Kaggle are collected but since the author is using google colab and due to lack of colab 12GB ram, only a few kinds of fruit and vegetables will be used. The author chooses a few kinds of fruit and vegetables based on common items which are easily found in daily life. 10, 20, 30, and 40 kinds of fruits will be collected for further analysis to see if the more dataset affects the time and object detection result. These images dataset are typically 100x100 pixels and easier for image processing especially since each image has a white background already consisting of 360-degree images.

Table 4.1: Fruits and Vegetables Kinds Dataset Table





































No.	Kinds	Fruits and Vegetables
1.	10	Apple Red Delicious, Blueberry, Cantaloupe 2, Cocos, Dragon Fruit, Guava, Lemon, Onion White, Orange, Tomato
2.	20	Apple Granny Smith, Apple Red, Apple Red Delicious, Blueberry, Cantaloupe 2, Cherry, Cocos, Corn, Dragon Fruit, Guava, Lemon, Limes, Mulberry, Onion White, Orange, Pepper Green, Pepper Red, Pepper Yellow, Tomato, Watermelon

Table 4.2: Kaggle Dataset Examples

Fruits and Vegetables Kinds		Images Dataset Examples				
<i>No.</i>	<i>Name</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>
1.	Apple Red					
						
2.	Apple Red Delicious					
						
3.	Cantaloupe 2					
						
4.	Corn					
						
5.	Guava					
						

As for the second section, selected fruits and vegetable images that were taken by the iPhone X camera will be used for the second section's main dataset which the author calls by input image one from the object's top view and the other one from the object's side view. The top view is to calculate the object's area and the side view to calculate the object's height where the author is going to multiply them in the end to find the object's mass. Images with different kinds, shapes, colors of fruit will be taken to identify the effectiveness of these programs. Each with a different image shooting distances which are 15, 17, 19, 21 cm. Each image taken by iPhone X consists of 3024 x 4032 pixels. As for the result, we can see these models can estimate an object's volume and mass accurately with particular shapes or colors or shooting distances that the author is going to analyze.

Table 4.3: Input Image Examples

Fruits and Vegetables Kinds		Image Distances					
<i>No.</i>	<i>Name</i>	<i>15</i>	<i>17</i>	<i>19</i>	<i>21</i>	<i>23</i>	<i>25</i>
1.	Apple Red Delicious						
							
2.	Lemon						
							
3.	Orange						
							

4.2. Dataset Preprocessing

4.2.1. *Input Image Preprocessing*

As for the first step, the author stored the dataset and input image on google drive, while the author run the code in google colab so integration between the two platforms is required. After both platforms have mounted, the next thing to do is to call and extract needed files to google colab. The next thing to do is to copy input images from the google drive path to the tmp folder in google colab with a determined name. In other words, the input images are copied and renamed. The reason behind this is for easier to call images on the model later which are inputtopview.jpg and inputsideview.jpg.

To improve the area calculation of the input images, as for the first step input images' background will be turned into white color. The reason why the author does not just take objects with an HVS background for the input image is that the object's shadow will be terminated and calculated for the area and produce inaccurate results considering taking an image without even slight shadow is hard. This step is used for both input image top view and side view.

The first step is to read the selected image which the first one is the top view image later same steps are used for the side view. After the author stored the image copy on a new variable in order to create an image mask. The goal of this model is to extract an object from its background accurately and the result is extracted object which has the most perfect edge. By converting image color space to another color it makes the object have significant color difference so the object's edge can be seen clearly and can be extracted precisely.

The image needs to be transformed into HSV first to have a clear color difference between the object and the background. After HSV, input images are turned into RGB and then grayscale to have an increased sight of color difference. The output has a slight difference by using HSV the object tends to have a more precise edge cut. Afterward, gaussian blur is added to give a blur effect where it will reduce. After gaussian blur, the object's image requires to be eroded because by using gaussian blur only and canny edge detection, the object output has a black border around the cropped object as the effect of the blurred edge. Next, after the object has successfully eroded, the object's edge is obtained by using Canny edge detection. Followed by dilation either because by using canny only, a particular part of the object gets cut off too so dilation increases the preciseness of the object's edge.

Afterwards the result will be transformed to a threshold. By using Otsu's Thresholding, the optimal global threshold value will be determined from an image histogram. Since most of the objects tend to have elliptical/circular shapes, then an elliptical/circular kernel is needed here cv2 is provided a structuring element function instead of using NumPy which has a rectangular-shaped kernel. After by using closing from cv2 morphologyEx function which consists of a dilation followed by erosion in order to remove noises on the object. The result is still in edge form where only the edge line has a different color from the background like in Figure 1, to be used as a mask, it needs to have a different color inside the edge which is the object so looping here and replacing pixel is needed where pixel inside the edge has a bright color and the background have a dark color like in Figure 2 as the result.

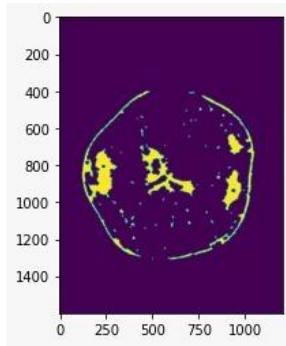


Figure 4.1: Data Preprocessing Result

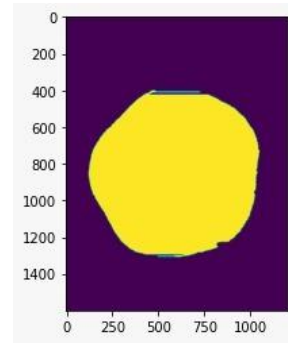


Figure 4.2: Mask Transformation

The following result will be masked into the input image so only the interesting part of the image is left and the rest will be in white color. After the image with white background is saved in the tmp folder with jpg format so that the next step is to replace all white pixels with transparent can be done easily by opening the image and converting to RGBA followed by converting the list into a buffer then recreating the image. Last is to save and show the image result. Result image is saved in the tmp folder provided with google colab with a determined name for easier to call later which is topviewextractedtr.png. Exact same steps happen for the side view image either but with a different saved image result name which is sideviewextracted.jpg for the side view image with white background and sideviewextractedtr.png for a transparent background and both of them in the tmp folder too.

4.2.2. Dataset Preprocessing

After we have done the input images preprocessing, it's the object's dataset turn which is extracted from the google drive path that has been mounted before to the provided tmp folder with "Training" as for the subfolder name for easier callables that consist of sub

subfolders with each fruits and vegetables kinds name as the sub subfolders name. After the zipped dataset has been successfully extracted, the next step is to have a side view image for the fruits and vegetables identification by adding new sub subfolders 'Input Image' inside the folder Training. Inside of the sub subfolders 'Input Image' where we place the side view of the input image by using a copy file command from google drive path to the new path and rename it with 'inputimage.jpg'. For easier processing of the data, the input side view image is stored within the same subfolder with the dataset but with a different sub subfolder which is "Input Image".

To read all the files within the specified folder, the author uses the glob module provided by python and stores them in the tuple. The tuple consists of each image's dataset path and name. For easier readable format data frame from pandas is used. Here the author can see the path with the dataset name easier and eliminate irrelevant data paths.

The dataset consists of raw images data that needs to be preprocessed for an understandable format. Since the DBSCAN algorithm works best with dataset in coordinate form, to acquire coordinates, first all dataset images need to be transformed in hash form. Dataset that is used for this research consists of images. The author chose hash instead of calculating the image RGB or pixel value because similar images tend to have similar hash value unlike other options. Imagehash package is required to transform the images dataset to hash form where we need to install it first in google colab.

The author creates 2 functions, one to calculate the hash and the other one to calculate the distance from the hash. The first function name is hashes_calculation which the parameter will consist of an image path and to call whash (wavelet hashing) from imagehash which is stored in hashfunc variable. Wavelet hashing using DWT (Discrete Wavelet Transformation) instead of DCT (Discrete Cosine Transform) like other options and whash is famous for it's great result either. First, declare the used variable which is hashes and names. Next with looping of each image dataset so every image in the dataset folder can be calculated to hash form. Inside of the looping is to open the image and use the hashfunc to calculate image hash and store them in hashes variable while names variable stored each of images name, both in list form. As for the result, the function returns hashes and names.

After all the images dataset has successfully turned into hash form, distance matrix will be calculated. The next function is distance calculation to find the distance between each image. First to declare a 'matrix' variable which has a matrix form with the length of the column and rows is the same as the hash length. Afterward looping is required to access every

image hash in the list. By using combinations from python the loop can access every possible combination of the list so distances between each of two images can be found and stored in the matrix variable that we have declared earlier and return the matrix.

From here with the help of PCA from sklearn, coordinates of each image can be acquired by using PCA with components of 2 and the result ready to enter the DBSCAN algorithm. By using coordinates clustering and visualization can be done easier either.

4.3. DBSCAN

To enter the DBSCAN clustering algorithm, the dataset needs to be transformed into list form and normalized. DBSCAN algorithm depends on eps and minPts value where eps stands for distance measure to find neighborhood within a point and minPts stands for a minimum number of points to form a cluster. The first step in the DBSCAN algorithm is to initialize all data points as outliers. After, the algorithm needs to find the neighbors for each data point. The core point is initialized where the neighborhood point is equal to or greater than minPoints. From here, core points, border points, and outliers can be found.

The DBSCAN algorithm is started by creating an object class named DBSCAN where it can be called easier later. After as for the constructor which is init where this method is called when an object is created from the class which consists of initializing core and border points with an arbitrary value.

The first function is find_neighbour where this function will find all the neighbors which arbitrary points in the dataset have. In this function, the author uses Euclidean Distance to find the distance between two points. By looping it will access all the data points and calculate the distance between them. The function will return neighborhoods points that particular points have and the distance between them is less than equal to determined epsilon along with storing them in list form on variable points.

The second function is fit which is the core of DBSCAN where this function will generate clusters based on processed data earlier. After finding all neighbors which each data point has now it's time to find all the clusters which this dataset has. The first step is to create a list to store which cluster each of the data points has. Zero here stands for outliers so as mentioned before all data points are considered as outliers with cluster zero. Next, initializing core and border as an empty list variables to store core and border points are needed.

There are 3 important sections on this function. The first one is to call the function we have made earlier which is `find_neighbour`. With the help of looping, each point can find all of its neighborhood points. In the second section, looping is utilized to find the core and border points by if any point has more or equal to determined `minPts` then the point is considered as a core point if the point doesn't satisfy the requirement then the point is considered as border points/outliers. The third section is to cluster each of the data points. Here the author performs queue data structure and chooses Breadth First Search (BFS) to find each data point that has been separated by epsilon distance and all the indirectly reachable points, the neighborhood points of the core can be obtained. If a point is considered as a core then it will be labeled as a cluster and find all the neighborhoods of the point which the author has grouped with the previous section and so on with the next cluster with a different core point. Now all points have been assigned to a cluster or cluster 0 for the outliers. This function returns `point_labels` which consist of a list of cluster labels of each point and number of clusters.

The last section is to visualize the clusters. A Scatter plot from matplotlib will help to visualize all the data points. Each cluster will be color-coded here the author chooses to have 3 different colors to identify each of the data points where the cluster they belong to but all the outliers will be colored as black. As for the outcome, each image is a member of a cluster.

The final step is to create a DBSCAN object and to call all of the functions we have made before which are `DBSCAN.fit` and completed by `DBSCAN.visualize` to visualize the outcome easier.

After we find which cluster our input image belongs to, the rest of the results outside the input image's cluster can be dropped. Now the dataset used only consists of images within the same cluster with the input image. For easier to process the next algorithm input image is being put at the first row as being the reference to find other images which have the closest distance to the input image.

4.4. K-NN

After the dataset is filtered and set up, it's ready for k-NN training data. The first step is to preprocess the data again by using image hash to distance matrix and with the help of PCA, coordinate form is obtained. These dataset coordinates will be k-NN training data. k-NN's main job is to find the closest point with the input image. k-NN uses euclidean distance to calculate the distance between each point as for the k value to decide which points have the

closest relationship with the input image. The first function consists of Euclidean Distance formula calculation to calculate distance between two data points which return the final result of square root. With euclidean distance formula as the follows:

$$d(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

The second function is to get neighbors which have 3 params consisting of train data, test data or selected points (input image point), and a number of neighbors the author is looking for. First to initialize variables to save each of the distances that have been calculated between two points by looping and calling the euclidean_distance function that we have made earlier in list form starting from the closest one. Next is to initialize a new variable to store the final result which is based on the k value from the param that has been determined. Here the author chooses the k value of two where only two points will be stored which are the input image point as the reference and the closest point with the input image. With k value, the algorithm will determine what the object is based on the closest points with the input image point. The result of k-NN will determine the success of object detection.

After the object has successfully been detected, the result is matched with the density table. The density table consists of all fruits and vegetable densities in CSV form. Dataframe from pandas is used for easier to read and find the matched density of fruit with the object detection result. Now the object's density is obtained and required for volume estimation.

4.5. Object's Volume Estimation

Object volume is needed for mass estimation. Mass and volume of fruits and vegetables have a linear relationship with their volume. The bigger the object then the heavier the object is. To calculate the object's volume top view and side view of the object are needed. 2 input images consist of a top view image and a side view image will be used. Object's area calculation from top view image is calculated by using image processing.

In the first section, data preprocessing images with transparent backgrounds have been acquired. Now the author is going to use those preprocessed images for area calculation and height estimation.

As for the area calculation, first, we use skimage.io to read the preprocessed top view image which has a transparent background because skimage will read and display in RGB. Next, turn the image into RGB and blur the image. After masking the image so that the rest

have a black background and only the interesting part of the image is selected. Reasons why not just turn the background black not using the transparent background first because it won't have a clean cut on the object edges. It helps to remove unwanted noises and increase the precision of edge detection.

Next is to calculate the area by obtaining the height and width of the image first then convert the image to gray and threshold it. To calculate the area is to count the non-zero pixel of the image which is the object. Now by counting the non-zero pixels the top area has been calculated.

Thereafter is height calculation, the same as previous which the image background needs to be converted to black by reading the image using `skimage.io`, convert to gray, and blur to have a clean object with a black background. Next is to convert it to gray and have the threshold the same as previous but not to calculate area instead to draw bounding boxes around the non-zero pixel. By creating a rectangle bounding box, the height of the object can be acquired.

For the side view image, the same steps will be used. The step is a bit different, grayscale and gaussian blur is still required but the threshold is used to calculate the object height by using a bounding box of the object. The area calculation result needs to be multiplied by 100 for the result in centimeters. Object's height needs to be divided by 100 to get the result in centimeters.

Last but not least is to calculate the volume, the author will use the following formula since the object's area and height have been calculated.

$$Volume = Area \times Height$$

The area needs to be multiplied by 1000 to achieve the real size of the area because earlier calculations provide a decimal result and the height needs to be divided by 100 to achieve the real size of object height. Next is to multiply them we have the object's volume.

4.6. Object's Mass Estimation

Now the object's density and object's volume have been found. As for the final step, by using the following formula

$$M = \rho \times V$$

An object's mass can be calculated. M here stands for mass of the object, ρ is for object's density which is the result of object detection matching with the stored density table and V stands for volume. With these mentioned steps, the object's mass is estimated.

4.7. Analysis

Conducted analysis in order to measure the success of this study is about accuracy comparison between real mass and estimated mass. The degree comparison result stands for the closeness between an object's real mass and obtained mass. Accuracy degree formula as the following.

$$Error_{percentage} = \frac{(Real_{mass} - Obtained_{mass}) * 100}{Real_{mass}}$$

$$Accuracy_{percentage} = 100 - Error_{percentage}$$

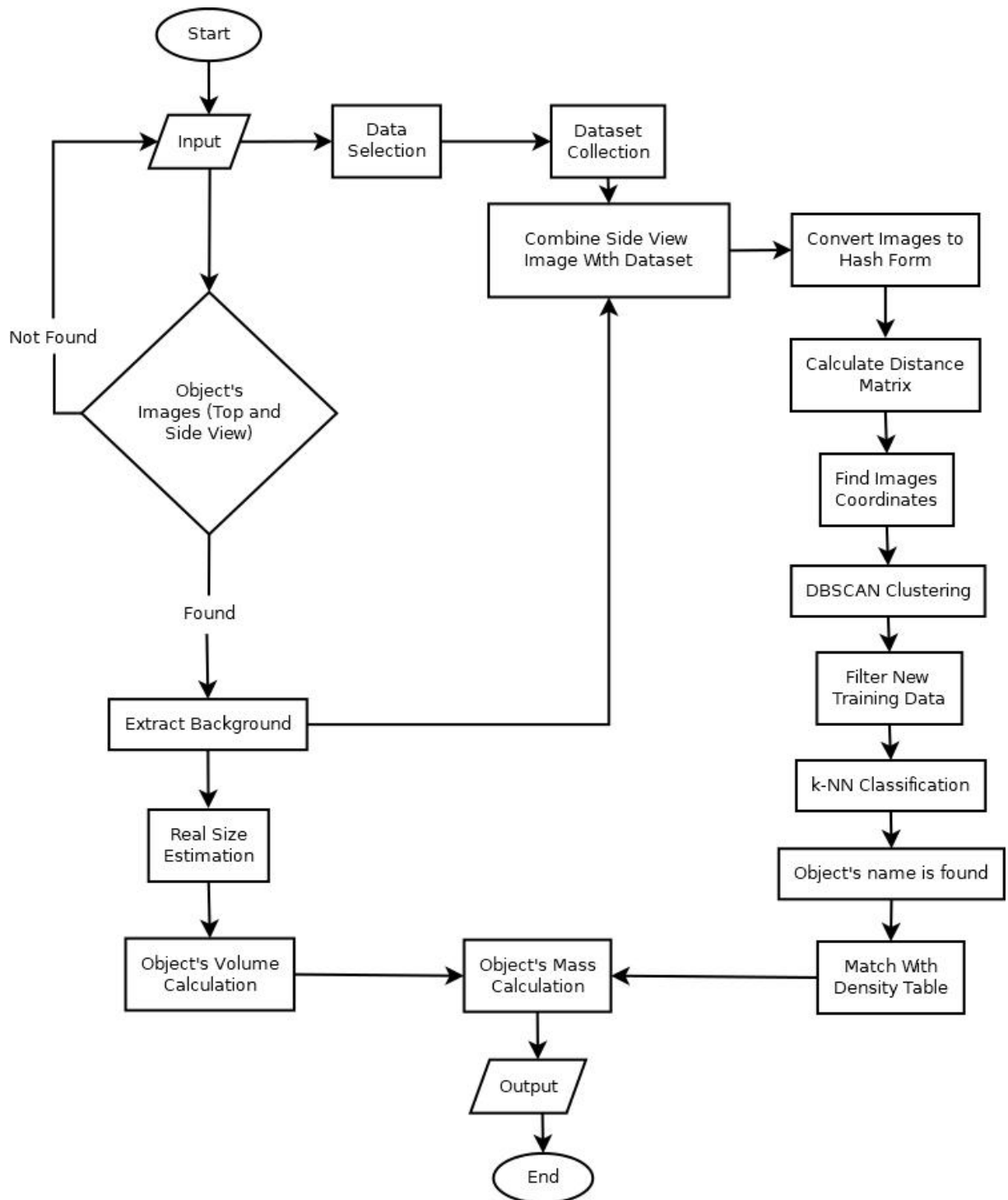


Figure 4.3: Real Mass Estimation Flowchart

CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1. Implementation

The computing language that is used in this research is Python and with the help of google colab to run the program, the author choose google drive to store the zipped dataset and input images. All the dataset and input images consist of raw data therefore data preprocessing is required so it can be used for the mass estimation model.

5.1.1. *Remove Input Image Background to Transparent*

```
1  img = cv.imread('/tmp/inputtopview.jpg', cv.IMREAD_UNCHANGED)
2  imagecopy = img.copy()
3
4  kernele = np.ones((8, 8), 'uint8')
5  kerneld = np.ones((4, 4), 'uint8')
6
7  filter = cv.cvtColor(img, cv.COLOR_BGR2HSV)
8  filter = cv.cvtColor(filter, cv.COLOR_HSV2RGB)
9  filter = cv.cvtColor(filter, cv.COLOR_RGB2GRAY)
10 filter = cv.GaussianBlur(filter, (17, 17), 17)
11 filter = cv.erode(filter, kernele, iterations=3)
12 filter = cv.Canny(filter, 23, 23)
13 filter = cv.dilate(filter, kerneld, iterations=3)
14
15 _, thresh = cv.threshold(filter, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
16 kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
17 mask = cv.morphologyEx(thresh, cv.MORPH_CLOSE, kernel, iterations=4)
18
19 data = mask.tolist()
20 sys.setrecursionlimit(10**8)
21 for i in range(len(data)):
22     for j in range(len(data[i])):
23         if data[i][j] != 255:
24             data[i][j] = -1
25         else:
26             break
27     for j in range(len(data[i])-1, -1, -1):
28         if data[i][j] != 255:
29             data[i][j] = -1
30         else:
31             break
32 image = np.array(data)
33 image[image != -1] = 255
34 image[image == -1] = 0imag
35
36 mask = np.array(image, np.uint8)
37 main = cv.bitwise_and(imagecopy, imagecopy, mask=mask)
38 main[mask == 0] = 255
39 cv.imwrite('/tmp/topviewextracted.jpg', main)
40 img = Image.open('/tmp/topviewextracted.jpg')
41 img.convert("RGBA")
42 datas = img.getdata()
43
```

```

44 datas2 = []
45 for item in datas:
46     if item[0]==255 and item[1]==255 and item[2]==255:
47         datas2.append((255, 255, 255, 0))
48     else:
49         datas2.append(item)
50
51 img.putdata(datas2)
52 img.save("/tmp/topviewextractedtr.png", "PNG")
53 plt.imshow(img)

```

The first line is to read the input image we are going to use for volume estimation. Same code works for input image top view and side view. As to discuss the code line, on the above code the author uses the top view image only. Second line is to copy the image so that it can be used later on line 37 to mask the image. The fourth and fifth is to initialize the kernel that is going to be used for erosion in line 11 and dilation in line 13.

Lines 7 to 13 is to convert color and filter the image to have a clean edge cut. As cv2 reads the image in BGR so the image needs to be converted to HSV to have a big color difference between the object and the background like in line 7. Next is to turn the image to RGB like in line 8 and to convert it again to grayscale. With grayscale, it increases the color difference and is easier for canny detection. There's a slight difference by using HSV or grayscale on but the edge can be cut more precisely with HSV. The next step is to add Gaussian Blur like in line 10 reasons behind this are to eliminate unwanted noise whether on the object or the background since the author calculates the area by object's image for the more accurate result is not counting the noise and eliminating them.

The next line is to use erosion with kernel (8,8) that have been mentioned before with 3 iterations by erode it helps the resulting image to not have a black border around the object as the result of Gaussian blur. Line 12 is canny edge detection to detect the object edge accurately. On line 13 is dilation with the kernel (4,4) and 3 iterations, dilation helps the object to have a better edge cut unlike without dilation where the edge cut is too small and untidy. After finishing to identify the precise edge, the result needs to be thresholded. The author uses Otsu's Thresholding like in line 15 and the next line, 16 is to determine the kernel. Here the structuring element is used to provide an elliptical/circular kernel instead of numpy which provides a rectangular shaped kernel since most of our object is centered and round shaped.

Next is to use closing from cv2 morphologyEx in line 17 which consists of dilation followed by erosion to remove noises and create a better edge result for the mask with 4

iterations. Next line which is 19 is to store the mask in list form and for the next line 20 set recursion limit is needed to have deeper recursion limit in order to avoid any crashes. Line 21 is for looping each line and line 22 with 27 is to loop every pixel color. The difference between those two lines is line 22 is started from the front pixel of the list while 27 is started from behind, the last pixel on the row. Inside of the second loop there's a decision part where if the loop meets 255 in the row member then the loop will break like in line 26 and 31 if not then it will be replaced by -1 like in line 24 and 29. So if in the row there's 255 the loop will break the same with the second row looping which is from behind so only the center pixel is left considering the result before is an edge detection, in other words only the lined edge of the object (255) in the picture. To have a white color (255) blocked inside the border to represent the object. On line 32 the result is saved into a new variable so that on line 33 if the image pixel is not -1 then it will be replaced by 255 which inside of the border all consist of 0 pixel and 255 represent the object. Line 34 is to replace all the -1 to 0 so only the background will have black color.

Line 36 is to insert the result before to mask variables in list form. Next in line 37 is to use bitwise AND from OpenCV to select the 'interesting part' which is object image only of the picture by using the mask we have made earlier. Next line 38 is to select all black colour which is the background to white colour. The result is saved to the tmp folder like in line 40 and for line 41 is to open the image and insert to img variable. Next line 41 is to convert the image to RGBA and Line 42 is to insert again img pixel in datas variable. Here on line 44 initializing the datas2 variable saves the list of new images later in list form. On line 45 until 49 is to change the picture background from white to transparent. Line 45 is looping to access all list that has been saved in datas variable. Inside of the loop is the decision which if the pixel has white colour 255,255,255 then it will be replaced with transparent which is 255,255,255,0 and 0 stands for alpha like in line 46 to 47. If not white colour then the pixel colour is stayed that way like in line 48 to 49. Next in line 51 is to have it in image form by using putdata() function and line 52 to save the image final result to the tmp folder. Line 53 is to show the final image result by using matplotlib. Same steps of the code works for the side view image too so here as the result the author has two transparent background images which are top view of the object and side view of the object.

5.1.2. Extract The Zipped Dataset

```
54 Files = namedtuple('File', 'name path')
55 dataset = []
56 p = Path('/tmp/Training/')
```

```

57 for item in p.glob('**/*'):
58     name = item.name
59     path = Path.resolve(item).parent
60     dataset.append(Files(name, path))
61
62 dataset

```

Line 54 is to declare a variable to store the extracted result in tuple form with named fields. The result later will be stored in the Files variable. Next is to declare a variable dataset in line 55 to store each of the tuples in list form. Line 56 is to store the selected path where the dataset is going to be extracted which is '/tmp/Training/'. As for accessing zipped folders and subfolders, looping and glob is utilized in line 57. Inside of the looping, line 58 is to store each of the image names in the 'name' variable next line 59 is to store the image path in the 'path' variable. Last in line 60, after the name and path of the images are obtained, both of them are stored in a dataset variable which has been declared in line 55 consisting of file tuples which are image name and path. Line 62 is to print the dataset variable to show the extracted result.

5.1.3. *Image Hash*

```

63 def hashes_calculation(files, hashfunc=imagehash.whash):
64     hashes, names = [], []
65     for i, name in enumerate(files):
66         try:
67             img = Image.open(name)
68             hash = hashfunc(img)
69             hashes.append(hash)
70             names.append(name)
71         except:
72             pass
73
74     return hashes, names

```

As to calculate the images hash value the author built a function which called hashes_calculation. Line 63 is to initialize the function with two params which are files to get the image path and hashfunc to store the built in function from imagehash which is whash (wavelet hashing). Inside of the function first is to initialize variables that are going to be used to store the result which are hashed and names like in line 64. Looping is required to access each of the images and enumerate them in line 65. Line 66 and 71 is for try and except where try is to test a block of code while except for handles the error. Inside of the try, line 67 is to open the image with it's particular path and stored in img variable. Next is to hash the image with the help of hashfunc which consists of imagehash from line 63 and stored it in a hash

variable. Followed by line 69 keeping the hash result in hashes variable and image's name in names variable in line 70. Line 72 is pass to avoid any error and the final step is to return the result which are hashes and names in line 74.

5.1.4. Distance Matrix

```
75 def distances_calculation(hashes):
76     matrix = np.zeros((len(hashes), len(hashes)))
77     for i, j in combinations(range(len(hashes)), 2):
78         dist = hashes[i] - hashes[j]
79         matrix[i, j] = matrix[j, i] = dist
80     return matrix
```

After finishing with hash, now is to calculate the distance between each of the images which the result will be in matrix form. First is to initialize the function like in line 75 which is distances_calculation and one parameter which is hashes where the list of hash values that have been calculated is stored. In line 76, 2 dimensional array both with the length of hashes list is initialized and stored in matrix variable. Followed by looping in line 77 to access every two images with combinations to help obtain every possible image couple. Inside of the loop is to calculate the distance of each image by subtracting hash value of both the images and storing it in dist variable like in line 78. Afterwards, line 79 is to insert the result in the matrix that has been made earlier. Last is to return the result of this function which is the matrix consisting of all distances between two images where can be seen in line 80.

5.1.5. Coordinates Form

```
81 pca = PCA(n_components=2)
82 dist2d = pca.fit_transform(matrix)
```

Now we have each image's distance in matrix form, to enter DBSCAN coordinates dataset is required. Besides it is easier for DBSCAN to process, it's easier to visualize either. To turn the distance matrix to coordinates a help from PCA is required. In line 81 is to set up the PCA with total components of 2 because of 2 dimensional vectors which are x and y and stored in the pca variable. Next, line 82 is to call the PCA built in function which is fit_transform to transform the distance matrix to coordinates and save in dist2d variable.

5.1.6. DBSCAN Algorithm

Built DBSCAN Object

```
83 class DBSCAN():
84     def __init__(self):
85         self.core = -1
```

```
86         self.border = -2
```

Now the dataset has been transformed into coordinates. This section will discuss how to build the DBSCAN model. The first step is to create a DBSCAN object like in line 83 by initializing the DBSCAN class. Next followed by line 84 which is initializing a DBSCAN constructor. In line 85 is for initializing core and line 86 border points both with arbitrary value.

Find Neighbour Function

```
87     def find_neighbour(self, data, point_id, eps):
88         points = []
89         for i in range(len(data)):
90             # Euclidian distance
91             if np.linalg.norm([a_i - b_i for a_i, b_i in zip(data[i], data[point_id])]) <= eps:
92                 points.append(i)
93         return points
```

After initializing the constructor, it's time to build a function here in line 87 the author built a `find_neighbour` function to find neighbours of each data point with 4 params which are `self` for represent the instance of the DBSCAN class, `data` for each of the dataset coordinates and `point_id` to represent each of the dataset id, and `eps` where to determine how close the distance of a point to be considered as neighbour. In this function Euclidean Distance is featured in order to find distances between two points. Inside of the function, first is to initialize the `points` variable where the result is going to be stored in list form that can be seen in line 88. In line 89, to access every data point, iteration through every coordinate is required. Inside of the loop in line 91 is the implementation of Euclidean Distance formula which if the result is less than equal to `eps` it's considered as it's point neighbour. The point will be stored in `points` variable like in line 92 and Last line 93 is the function return which is a `points` variable that consists of the final result.

Fit Function

```
94     def fit(self, data, Eps, MinPts):
95         point_label = [0] * len(data)
96         point_count = []
97
98         core = []
99         border = []
100
101         for i in range(len(data)):
102             point_count.append(self.find_neighbour(data, i, Eps))
103
104         for i in range(len(point_count)):
105             if (len(point_count[i]) >= MinPts):
106                 point_label[i] = self.core
107                 core.append(i)
108             else:
```

```

109         border.append(i)
110
111     for i in border:
112         for j in point_count[i]:
113             if j in core:
114                 point_label[i] = self.border
115                 break
116
117     cluster = 1
118
119     for i in range(len(point_label)):
120         q = queue.Queue()
121         if (point_label[i] == self.core):
122             point_label[i] = cluster
123             for x in point_count[i]:
124                 if (point_label[x] == self.core):
125                     q.put(x)
126                     point_label[x] = cluster
127                 elif (point_label[x] == self.border):
128                     point_label[x] = cluster
129             while not q.empty():
130                 neighbors = point_count[q.get()]
131                 for y in neighbors:
132                     if (point_label[y] == self.core):
133                         point_label[y] = cluster
134                         q.put(y)
135                     if (point_label[y] == self.border):
136                         point_label[y] = cluster
137             cluster += 1
138
139     return point_label, cluster

```

Calculating distance function has been done now it's turn to build a clustering function which is fit function. First is to initialize a Fit function like in line 94 which consists of 4 params which are self, data, eps, and minPts. Self is to represent the instance of the DBSCAN class, data for the coordinates dataset, eps will be determined how close the distance to be considered as a neighbour, and minPts is for minimum data points to be considered as a cluster. Next line 95 is initializing the point_label variable of the data points cluster labels which are in list form and [0] here stands for cluster 0 or outliers in other words all of the data points considered as outliers first. In line 96 point_count variable is initialized to store the find neighbours function result earlier. After, line 98 is to initialize the core variable to store all the core points in list form and line 99 is to initialize border points to store all the border points in list form either.

All the required variables have been initialized. Next line 101 is to iterate through all the dataset and line 102 inside of the loop is the function we have made earlier find_neighbour to find all the neighbours of particular data points and the result is stored in the point_count variable.

Each point has its own neighbours but it needs to be defined which points are core points, border points, and outliers. It starts with iteration to access every data point like in line 104. Inside of the loop is decision therefore in line 105 if the point has more than equal to minPts which minimum points than it's point is considered as core point like in line 106 and line 107 is to store core point in core variable. Then if the point doesn't fulfill the requirements then the point goes into the else in line 108 and in line 109 the point is stored in the border variable. Followed by another iteration in line 111 for every point in the border list which is to confirm whether the point is border point or core point. Line 112 is another iteration to access it's particular point in point_count variable. Inside or two iteration in line 113 is a decision if the particular point in the core list then it's point_label is replaced with a border like in line 114 and line 115 is to break the loop.

After determining which point is the core point and which point is the border point now it's turn to form the clusters. In line 117 cluster variable is initialized to count the number of clusters formed. The count starts from one. Followed by iteration in line 119 to access every point label. In line 120 is to initialize q which consists of queue built in function Queue() which is a constructor for a FIFO queue. Next is line 121, it's a decision if the particular point is in the core list then the point is in the new cluster like in line 122 by replacing the particular point label. In line 123 there is iteration for every point group in point_count. Followed by decision in line 124 if a particular point label is considered as a core point then in line 125 there is put queue function to put an item into the queue until a free slot is available before the item is added. Before, all point labels are 0 here in line 126 the point label of a particular point is replaced with the cluster number. Next is line 127 for the else if decision, if a particular point is in the border point list then in line 128 a particular point label is replaced with cluster number.

Next line 129 is a while loop as long as the queue is not empty then in line 130 there's get queue function to wait until an item in queue is available. Point count of a particular point is stored in the neighbors variable. In this loop is to check every point of the particular point neighbour. In line 131, for every point in neighbours variable, if the point is a core point like in line 132 then the point label of it's point is replaced with the cluster number as in line 133. Next line 134 is to be put in queue again. Followed by line 135 if the particular point is border point then the point label of the particular point is replaced with cluster number either as in line 136. Line 137 is to initialize the cluster number by adding plus one for every iteration. Last line in this function is line 139 to return the point label and cluster this function has

formed. With this function Breadth First Search is formed by every node in n level then n+1 level and so on.

Visualization Function

```
140 def visualize(self, data, cluster, clusters_count):
141     N = len(data)
142
143     colors = np.array(list(islice(cycle(['#FE4A49', '#2AB7CA']), 3)))
144
145     for i in range(clusters_count):
146         if (i == 0):
147             color = '#000000'
148         else:
149             color = colors[i % len(colors)]
150
151     x, y = [], []
152     for j in range(N):
153         if cluster[j] == i:
154             x.append(data[j, 0])
155             y.append(data[j, 1])
156     plt.scatter(x, y, c=color, alpha=1, marker='.')
157     plt.show()
```

To ease reading the clustering result, visualization is needed. Here the visualization function will be discussed. First is to initialize the visualization function like in line 140 which consists of 4 params. Self, to represent the instance of the DBSCAN class, data is for the dataset, cluster for the point_labels by fit function, and clusters_count is to get the cluster number that's been formed by fit function. Next in line 141 there is N variable to store the length of the dataset. By line 143 is to store various colors that are going to be used to represent the clusters by adding them into an array and stored in *colors* variable. Next in line 145 an iteration is used to access every cluster. Inside of the loop is a decision part in line 146 if the cluster is 0 then in line 147 is to represent the cluster color which is black and where cluster 0 is an outlier. Else in line 148, if the cluster is not 0 then in line 149 the cluster color is represented based on colors that have been initialized earlier in line 143.

In line 151, x and y are initialized in order to store the dataset coordinates later. After, line 152 is an iteration which is needed to access every dataset stored in N. Inside of the loop there's a decision that can be seen in line 153 for a particular cluster then x coordinates of the data points is stored in x variable like in line 154 and y coordinates of the data points is stored

in y variable like in line 155. Line 156 is to form the scatter plot of the clustering result by matplotlib. Last, line 157 is to show the visualization result.

DBSCAN Implementation

```
158 df = pd.read_csv("/content/ProjectDataset/MyDrive/coordinates.csv")
159 dataset = df.astype(float).values.tolist()
160
161 X = StandardScaler().fit_transform(dataset)
162 DBSCAN = DBSCAN()
163 point_labels, clusters = DBSCAN.fit(X, 0.1, 4)
164 print(point_labels, clusters)
165 DBSCAN.visualize(X, point_labels, clusters)
```

In this section is where all of the DBSCAN functions we have made are called. First line 158 is to read our coordinates dataset in csv form for easier callable and stored in df variable. In line 159, the dataset from df is transformed into list form and stored in the dataset variable. In line 161 the dataset needs to be normalized in order to have new values but still maintain general distribution and ratios among the data source while still keeping values within the scale of all the numeric columns in the model to avoid problems of great difference values to scale in the modelling. After the dataset has been normalized, now begin the DBSCAN clustering. In line 162 the DBSCAN object is created and stored in the DBSCAN variable. Next line 163 is to call the fit function with 3 parameters to be sent which are the normalized dataset, determined eps and minPts to be saved into point_labels and clusters variable. Line 164 is to print the point_labels and clusters result. Last is line 165 to call the visualization function which consists of 3 parameters X stands for the normalized dataset, point_labels and clusters from the fit function result.

5.1.7. Input Image Cluster

```
166 pntlbl = int(input_img['Point Labels'])
167 filter_data = all_files[all_files['Point Labels'] == pntlbl]
168 filter_data
```

This section is to filter only images that belong to the same cluster as the input image is taken to become k-NN training data, the other cluster is dropped. Point Labels represent the cluster number. First is to find what the input image cluster number like that have been done in line 166 and the result is stored in pntlbl variable. Next is to filter the data which is to find all the dataset images that have the same number of point labels as the input image like in line 167, the result is stored in filter_data variable and line 168 is to present them.

5.1.8. *k*-NN Algorithm

Euclidian Distance Function

```
169 def euclidean_distance(vec1, vec2):
170     distance = 0.0
171     for i in range(len(vec1)-1):
172         distance += (vec1[i] - vec2[i])**2
173     return sqrt(distance)
```

After the data has already been filtered, only images with the same cluster with the input image are left. Here the data becomes *k*-NN training data and the goal is to find the closest point with the input image. To build the *k*-NN algorithm started by building the calculation. To calculate the distance between two vectors the author uses Euclidean Distance. Started by initializing the euclidean distance function which has 2 params *vec1* and *vec2* where *vec1* stands for the input image coordinate that becomes the reference and *vec2* stands for all the training dataset coordinates that can be seen in line 169. Next is to initialize the distance variable like in line 170 that is going to be used. In line 171 an iteration is used in order to iterate every item in the test dataset but the test image that is used in this model is 1 so only 1 iteration. Inside of the loop there is the Euclidean Distance formula in line 172 which is to subtract vector 1 (test image) with vector 2 (training image) and square them then the result is stored in distance variable. This function returns the square root of the distance variable which consists of the square result like in line 163.

Get Neighbours Function

```
174 def get_neighbors(train, test_row, num_neighbors):
175     distances = list()
176     for train_row in train:
177         dist = euclidean_distance(test_row, train_row)
178         distances.append((train_row, dist))
179     distances.sort(key=lambda tup: tup[1])
180     neighbors = list()
181     for i in range(num_neighbors):
182         neighbors.append(distances[i][0])
183     return neighbors
```

The distance calculation function has been done now it's turn to make a function to find the closest neighbours of the input image. In line 174, the *get_neighbours* function is initialized with 3 params, those are *train* for the dataset, *test_row* for the reference which is input image, and *num_neighbors* is for the *k* value. In line 175 *distances* variable is declared to save data in list form. Followed by iteration for every item in the train dataset like in line 176. Inside of the loop can be found a *euclidean_distance* function. Here in line 177 the

euclidean function is called to count distance between reference coordinate and particular train dataset coordinate, the result is stored in dist variable. Next is to store the train coordinates and it's distance in distances variable like in line 178. Here in line 179 is to sort the result stored in distances variable based on the closest one with the input image. Afterwards another variable is initialized in line 180 which is neighbours to store data in list form. Another iteration is used in line 181 based on the k value that has been determined. Inside of the loop is line 182 where the distance variable is being called. This iteration is used in order to find the closest distance based on k value. In this model the author uses 2 k values so only 2 iterations which is the input image and one point that have the closest distance with the input image. Last line in this function is 183 where this function returns the neighbours variable consisting of the final result.

K-NN Implementation

```
184 n = []
185 dataset = dist2d2
186 neighbors = list(get_neighbors(dataset, dataset[0], 2))
187 n = n + neighbors
188 print (n)
```

After finishing building all the required functions, the final step is to call the get_neighbours function that has been built earlier. First in line 184 is to declare n variable to store data in list form. Next, line 185 is to store the normalized dataset in dataset variable. After, in line 186 the get_neighbors function is called and the result is stored in neighbors variable in list form. In line 187 the neighbors variable is being added into n variable that has been made earlier. Line 188 is to display the final result which consists of the reference coordinate which is the input image and a coordinate of the closest point with the input image. Now we have the final result about what the input image object is based on what object has the closest distance with the input image.

5.1.9. Get Object's Density

```
189 final_dense = dens[dens['Fruit'] == fr]
190 density = float(final_dense['Density (g/cm3)'])
```

After the object has been successfully identified, the object's density is required. Start by finding the object's density from the stored density which has the same object's name. Line 189 is to select the row which has the same object's name and store it in the final_dense

variable. Line 190 is to select the density of the selected object and store it in the density variable. Now the object's density has been acquired.

5.1.10. Volume Calculation

Black Background Transformation

```
191 image = skimage.io.imread(fname='/tmp/topviewextractedtr.png')
192 blur = skimage.color.rgb2gray(image)
193 mask = blur < 0.98
194 blackbg = np.zeros_like(image)
195 blackbg[mask] = image[mask]
196 skimage.io.imshow(blackbg)
```

In order to estimate the object's volume, area calculation and object's height is required. Since all the input images have a transparent background, both of the input images, top view and side view need to be transformed to have a black background for easier calculation. Start by reading the selected image, for the first section the input image is the object's top view and stored in the image variable. Here in line 191 skimage is used in the reason skimage reads images in RGB. Next is to transform the image to grayscale and store the result in a blur variable like in line 192. After, in line 193 blurring effect is being added and the result stored in the mask variable. In line 193, blackbg is being declared consisting of black color image with the same feature as the image. Next is both of the black background and image is being masked based on the transformed blurred image resulting in combined result with only the background being changed into black. With black background threshold for the next section can be done easier and more precisely. In this section only the top view image is being presented, same steps happen for the side view image either.

Area Calculation

```
197 img = blackbg
198 height = img.shape[0]
199 width = img.shape[1]
200 gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
201 ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY)
202 plt.imshow(thresh)
203 count = cv2.countNonZero(thresh)
204 area = count*0.14*0.14/(width*height)
205 print(area)
```

Object's area calculation is performed by calculating the non zero pixel. First step is by reading the object's top view image that has been transformed to black background like in line 197. Next, in line 198 image height is being calculated and in line 199 image width is being calculated. Afterwards the image is transformed into grayscale like in line 200. After it

is finished turning into grayscale and stored into a gray variable, the image needs to be thresholded like in line 201. Turning a black and white image and stored in the thresh variable. Next in line 202 is to show the result of threshold. To calculate the area is to count the non zero pixel considering only the image background is black or zero so the object or non zero pixel is being calculated like in line 203. The calculated result is being stored in the count variable. In line 204, the area is calculated by multiplying count with 0.14 which represents the ground area or dimensions in meters and divided by multiplied images height and width, the result is stored in the area variable. Line 205 is to show the result.

Height Calculation

```

206 img = blackbg2
207 gray2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
208 ret, thresh2 = cv2.threshold(gray2,0,255,cv2.THRESH_BINARY)
209 plt.imshow(thresh2)
210
211 image = skimage.io.imread(fname='/tmp/sideviewextractedtr.png')
212 x,y,w,h = cv2.boundingRect(thresh2)
213 cv2.rectangle(image, (x, y), (x + w, y + h), (36,255,12), 2)
214 cv2.putText(image, "w={},h={}".format(w,h), (x,y) -
    10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (36,255,12), 2)
215 plt.imshow(image)
216 print(h)

```

Next step is to calculate the object's height. Same steps as been mentioned before where the side view image needs to be transformed to have a black background either. After the side view image has a black background, it is stored in an img variable like in line 206. Next in line 207 the image is transformed into grayscale, the result stored in gray2 variable and in line 208 the image is thresholded, the result stored in thresh2 variable. Line 209 is to show the threshold result. Same steps with the top view image, the top view image uses the non zero pixels to calculate the object's area but here the side view image uses non zero pixels to draw a bounding box and estimate the object height.

In line 211 the input image is being read and stored to the image variable. Side view input image is used as a background where the bounding box is drawn. Next line 212 is to create a bounding box according to non zero pixels from the threshold result where x,y is the top left coordinate of the rectangle and w,h is it's width and height. In line 213 a rectangle bounding box with green color is being drawn. Line 214 is to put text on the top left of the bounding box. Next, line 215 is to show the image with the bounding box result consisting of the object's height and width. Last is line 216 is to print the object's height. Object's height has been found.

Objects Volume

```
217 V = round((area*10000))*round((h/100)/2)
```

Object's density and height has been obtained and by multiplying them, volume can be acquired. In order to have a real size estimation, the object's area needs to be multiplied by 10000 and the object's height needs to be divided by 100 and again by 2 because the object's real size height is half of the result. In line 217, by multiplying area and height the result is the object's volume and object's volume is being stored in the V variable.

5.1.11. Object's Mass Estimation

```
218 M = density * V  
219 round(M)
```

Last but not least is to calculate the goal of this model which is the object's mass. Object's density and object's volume has been acquired, by multiplying them the object's mass can be found. In line 218 the object's mass is being calculated and the result is stored in the M variable. Followed by line 219 where the result is being rounded. Object's mass has been found.

5.2. Results

5.2.1. DBSCAN Algorithm

Table 5.1: DBSCAN Time and Cluster Test Result 1

No.	Eps, minPts		Training Data 10		Training Data 20	
			Time	Cluster	Time	Cluster
1.	0,1	2	5m 4s	128	22m 35s	116
2.	0,1	3	5m 2s	99	22m 38s	81
3.	0,1	4	5m 3s	90	22m 39s	61
4.	0,1	5	5m 3s	80	22m 37s	50
5.	0,1	6	5m 0s	69	22m 23s	47

Table 5.2: DBSCAN Time and Cluster Test Result 2

No.	Eps, minPts		Training Data 30		Training Data 40	
			Time	Cluster	Time	Cluster
1.	0,1	2	47m 59s	105	1h 39m 5s	94
2.	0,1	3	46m 15s	82	1h 38m 41s	71
3.	0,1	4	46m 6s	70	1h 28m 14s	65
4.	0,1	5	46m 39s	58	1h 26m 40s	58
5.	0,1	6	46m 34s	59	1h 26m 30s	54

5.2.2. Object Identification

10 Kinds in Training Data (4.802 images)

Table 5.3: Object Identification Result of 10 Fruit Kinds

No,	Objects Name	Identified (Eps ; Minpts)				
		0,1 ; 2	0,1 ; 3	0,1 ; 4	0,1 ; 5	0,1 ; 6
1.	Apple Red Delicious	Yes	Yes	Yes	Yes	Yes
2.	Blueberry	Yes	Yes	Yes	Yes	Yes
3.	Cantaloupe 2	Yes	Yes	Yes	Yes	Yes
4.	Cocos	Yes	Yes	Yes	Yes	No
5.	Dragon Fruit	Yes	Yes	Yes	Yes	Yes
6.	Guava	Yes	Yes	Yes	Yes	Yes
7.	Lemon	Yes	Yes	Yes	Yes	Yes
8.	Onion White	Yes	Yes	Yes	Yes	Yes
9.	Orange	Yes	Yes	Yes	Yes	Yes
10.	Tomato	Yes	Yes	Yes	Yes	Yes

20 Kinds in Training Data (10.207 images)

Table 5.4: Object Identification Result of 20 Fruit Kinds

No,	Objects Name	Identified (Eps ; Minpts)				
		0,1 ; 2	0,1 ; 3	0,1 ; 4	0,1 ; 5	0,1 ; 6
1.	Apple Granny Smith	Yes	Yes	Yes	Yes	Yes
2.	Apple Red	No	No	No	No	No
3.	Blueberry	Yes	Yes	Yes	Yes	Yes
4.	Cantaloupe 2	Yes	Yes	Yes	Yes	Yes
5.	Cocos	Yes	Yes	Yes	No	No

6.	Corn	Yes	Yes	Yes	Yes	Yes
7.	Dragon Fruit	Yes	Yes	Yes	Yes	Yes
8.	Lemon	Yes	Yes	Yes	Yes	Yes
9.	Orange	Yes	Yes	Yes	Yes	Yes
10.	Pepper Red	No	No	No	No	No

5.2.3. Object's Area

Apple

Table 5.5: Mass Estimation Result of 6 Apples

No.	Fruits	Distances	Estimated Area (cm)	Estimated Height (cm)	Estimated Mass (g)	Real Mass (g)
1.	Apple 1	15 cm	23	9	199	139
		17 cm	21	8	161	139
		19 cm	17	8	131	139
		21 cm	16	6	92	139
		23 cm	12	6	69	139
		25 cm	11	5	53	139
2.	Apple 2	15 cm	45	11	475	161
		17 cm	36	9	311	161
		19 cm	28	8	215	161
		21 cm	23	7	155	161
		23 cm	18	6	104	161
		25 cm	16	6	92	161
3.	Apple 3	15 cm	41	11	433	160
		17 cm	34	9	294	160
		19 cm	25	8	192	160
		21 cm	21	7	141	160
		23 cm	16	6	92	160
		25 cm	11	6	63	160
4.	Apple 4	15 cm	24	8	184	164
		17 cm	24	9	207	164
		19 cm	17	8	131	164
		21 cm	16	7	108	164
		23 cm	12	6	69	164
		25 cm	12	6	69	164

5.	Apple 5	15 cm	30	6	173	156
		17 cm	21	7	141	156
		19 cm	16	7	108	156
		21 cm	14	6	81	156
		23 cm	13	6	75	156
		25 cm	11	6	63	156
6.	Apple 6	15 cm	26	9	225	159
		17 cm	17	9	147	159
		19 cm	16	8	123	159
		21 cm	14	7	94	159
		23 cm	13	13	162	159
		25 cm	11	6	63	159

Lemon

Table 5.6: Mass Estimation Result of 6 Lemons

No.	Fruits	Distances	Estimated Area (cm)	Estimated Height (cm)	Estimated Mass (g)	Real Mass (g)
1.	Lemon 1	15 cm	27	12	295	107
		17 cm	21	11	211	107
		19 cm	19	9	156	107
		21 cm	14	8	102	107
		23 cm	13	7	83	107
		25 cm	11	6	60	107
2.	Lemon 2	15 cm	30	12	328	113
		17 cm	24	10	219	113
		19 cm	17	9	140	113
		21 cm	13	8	95	113
		23 cm	13	7	83	113
		25 cm	9	7	57	113
3.	Lemon 3	15 cm	35	12	383	107
		17 cm	25	10	228	107
		19 cm	18	8	131	107
		21 cm	13	7	83	107
		23 cm	10	7	64	107
		25 cm	8	6	44	107

4.	Lemon 4	15 cm	23	11	231	110
		17 cm	19	9	156	110
		19 cm	14	8	102	110
		21 cm	12	8	88	110
		23 cm	10	7	64	110
		25 cm	8	6	44	110
5.	Lemon 5	15 cm	25	10	228	108
		17 cm	18	9	148	108
		19 cm	14	7	89	108
		21 cm	12	7	77	108
		23 cm	10	6	55	108
		25 cm	8	6	44	108
6.	Lemon 6	15 cm	25	10	228	108
		17 cm	18	9	148	108
		19 cm	14	7	89	108
		21 cm	12	7	77	108
		23 cm	10	6	55	108
		25 cm	8	6	44	108

Orange

Table 5.7: Mass Estimation Result of 6 Oranges

No.	Fruits	Distances	Estimated Area (cm)	Estimated Height (cm)	Estimated Mass (g)	Real Mass (g)
1.	Orange 1	15 cm	53	10	546	237
		17 cm	37	8	305	237
		19 cm	32	7	231	237
		21 cm	28	6	173	237
		23 cm	22	5	113	237
		25 cm	20	5	103	237
2.	Orange 2	15 cm	68	10	700	250
		17 cm	47	8	387	250
		19 cm	35	7	252	250
		21 cm	30	6	185	250
		23 cm	26	6	161	250
		25 cm	22	5	113	250

3.	Orange 3	15 cm	64	10	659	237
		17 cm	49	8	404	237
		19 cm	39	7	281	237
		21 cm	32	6	198	237
		23 cm	25	6	154	237
		25 cm	23	5	118	237
4.	Orange 4	15 cm	58	9	538	209
		17 cm	39	7	281	209
		19 cm	33	6	204	209
		21 cm	27	5	139	209
		23 cm	20	5	103	209
		25 cm	20	5	103	209
5.	Orange 5	15 cm	38	9	352	205
		17 cm	28	8	231	205
		19 cm	27	7	195	205
		21 cm	26	6	160	205
		23 cm	18	5	93	205
		25 cm	19	5	98	205
6.	Orange 6	15 cm	53	10	546	236
		17 cm	36	8	297	236
		19 cm	31	7	224	236
		21 cm	28	6	173	236
		23 cm	23	5	118	236
		25 cm	19	5	98	236

Average Accuracy Result

Table 5.8: Average Accuracy of Apples

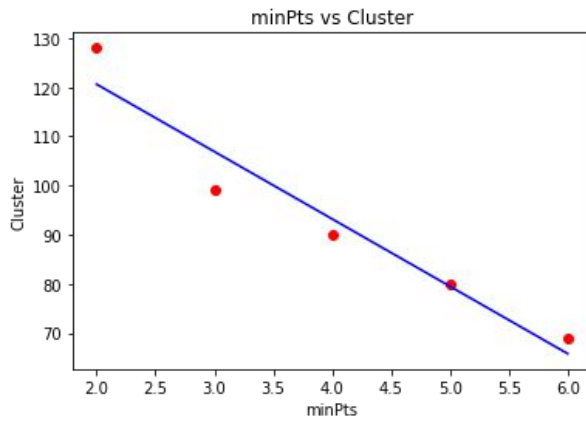
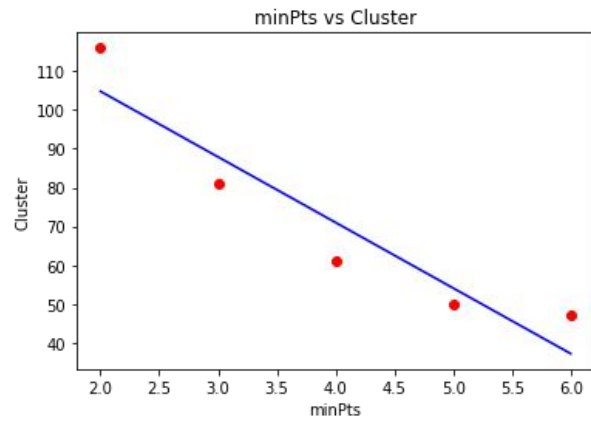
No.	Object's Name	Accuracy	No.	Objects's Name	Accuracy
1.	Apple 1	94,2%	4.	Apple 4	79,9%
2.	Apple 2	96,3%	5.	Apple 5	90,4%
3.	Apple 3	88,1%	6.	Apple 6	92,4%
				Average	90,2%

Table 5.9: Average Accuracy of Lemons

No.	Object's Name	Accuracy	No.	Objects's Name	Accuracy
1.	Lemon 1	95,3%	4.	Lemon 4	92,7%
2.	Lemon 2	84%	5.	Lemon 5	82,4%
3.	Lemon 3	77,6%	6.	Lemon 6	82,4%
				Average	85,7%

Table 5.10: Average Accuracy of Oranges

No.	Object's Name	Accuracy	No.	Objects's Name	Accuracy
1.	Orange 1	97,5%	4.	Orange 4	97,6%
2.	Orange 2	99,2%	5.	Orange 5	95,1%
3.	Orange 3	83,5%	6.	Orange 6	94,9%
				Average	94,6%

**Figure 5.1: Cluster and minPts Regression Linear 1****Figure 5.2: Cluster and minPts Regression Linear 2**

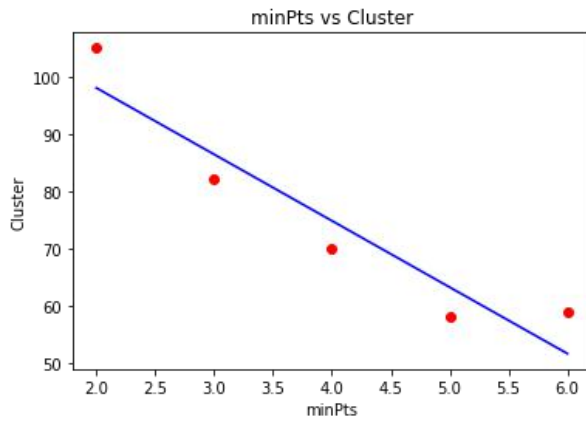


Figure 5.3: Cluster and minPts Regression Linear 3

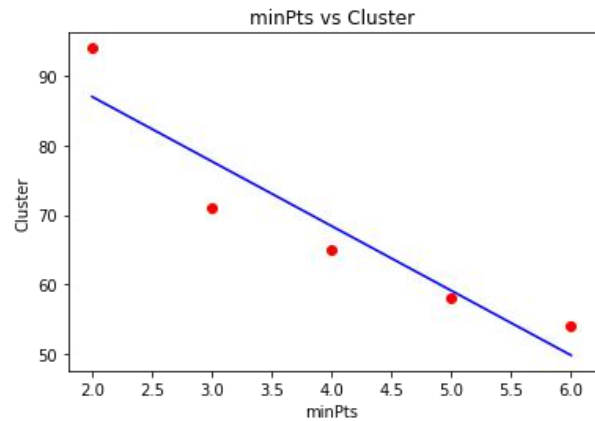


Figure 5.4: Cluster and minPts Regression Linear 4

The first test is the author uses 10, 20, 30, and 40 training data to measure time and cluster that has been formed by DBSCAN. eps stands for neighbourhood radius and minPts stands for minimum point to form a cluster. Here the author uses 0,1 for the eps to have a detailed result as every close point is counted. As for the result from Table 5.1 and 5.2, The higher the value of the minPts the less time it is required and less cluster is formed. Can be seen that minPts has a nonlinear relationship with time and cluster amount.

Next is to test the success of this algorithm by identifying objects. First test is held for 10 kinds of fruit each which consist of total of 4.802 images. Can be seen in Table 5.3 where all the object has successfully identified out of all the tests except Cocos with minPts 0,6 that misidentified as apple red delicious.

Another test is held with 20 kinds of fruit that consist of 10.207 images. And the result is not as good as the previous one, because although more data is loaded, each fruit has the same portion of train data. Out of 20 kinds of fruits there are 2 kinds that are not identified which are apple red and pepper red, besides the same as previous one this model fails in identifying cocos in high value of minPts which are 5 and 6.

Next is the mass estimation is tested within 3 kinds of fruit which are apples, lemons, and oranges each with 6 amounts of fruits. Out of 6 apples being tested, the closest one is the 2nd apple with 6 grams different and 96,3% accuracy. Next is the lemons, out of 6 lemons being tested the closest one with their real mass is the first lemon which has 5 grams difference and 95,3% accuracy with it's real mass. Next is the orange, here the 2nd orange being tested and the result is 2 grams different and 99,2% accuracy with the orange real mass.

Out of 18 fruits with distances of 15, 17, 19, 21, 23, and 25, most of the results achieve the highest accuracy in distance of 19cm between the camera and the object. Among all the highest results obtained, the author calculated the average accuracy with 90,2% for the apples, 85,7% for the lemons, and 94,6% for the oranges.

Figure 5.1 to 5.4 represent the relationship between minPts and the amount of cluster formed. Here minPts and clusters from 10, 20, 30, and 40 training data in table 5.1 and 5.2 are represented with Regression Linear graph and all the results are proved that minPts and the amount of clusters formed have a nonlinear relationship which the higher value of minPts the smaller amount of cluster formed.

CHAPTER 6

CONCLUSION

Based on the results of the conducted tests, this study verified that an object's mass can be calculated by using the object's image top view and side view. DBSCAN helps a lot in improving the accuracy of this model, in which the DBCAN cluster can successfully grouped the data based on the similarity, making k-NN easier in locating the nearest point neighbour. From previous tests being held, smaller minPts value proved in increasing the result accuracy which is reasonable in reason although the output consist of more clusters but each cluster tends to have a more accurate similarity. Moreover, minPts value has a nonlinear relationship with the amount of cluster formed which is proved by the analysis result and regression linear graph where the higher of minPts value the smaller amount of cluster formed.

Various distances yet affecting the result which can be seen in previous tests and 19cm is the best distance that achieves the highest accuracy among the others. Oranges have the best accuracy yet it's proven that this model works best in estimating the real size of objects with green color. This model will identify the object's edge more accurately when working with light color objects and black backgrounds because they tend to have a big color difference.

This mass estimation model accuracy average is 90,2%. This leads to the conclusion that the results have a good accuracy considering a small dataset and the author took the object's picture manually with a phone and ruler for the distance which increases the possibility of human error.

This study suffers from a small sample of data and manual measurement. Nevertheless this study's final result is promising and has laid a great foundation for further work. However further improvement is needed such as accurate measurement, image lightning, and wider dataset to achieve a higher accuracy .

REFERENCES

- [1] T. Standley, D. Chen, O. Sener, and S. Savarese, “image2mass: Estimating the Mass of an Object from Its Image,” p. 10.
- [2] K. Kollis, C. S. Phang, T. M. Banhazi, and S. J. Searle, “WEIGHT ESTIMATION USING IMAGE ANALYSIS AND STATISTICAL MODELLING: A PRELIMINARY STUDY,” *APPLIED ENGINEERING IN AGRICULTURE*, vol. 23, p. 7.
- [3] D. A. Konovalov, A. Saleh, D. B. Efremova, J. A. Domingos, and D. R. Jerry, “Automatic Weight Estimation of Harvested Fish from Images,” *arXiv:1909.02710 [cs, eess]*, Sep. 2019, Accessed: Sep. 01, 2021. [Online]. Available: <http://arxiv.org/abs/1909.02710>
- [4] L.-H. N. Nystad, “Automate Detection and Weight Estimation of Fish in Underwater 3D Images,” p. 56.
- [5] Sanchez, German, Ceballos Arroyo, Alberto, and Robles-Serrano, S., “Automatic measurement of fish weight and size by processing underwater hatchery images,” *Engineering Letters*, vol. 26, pp. 461–472.
- [6] J. A. Lines, R. D. Tillett, L. G. Ross, D. Chan, S. Hockaday, and N. J. B. McFarlane, “An automatic image-based system for estimating the mass of free-swimming fish,” *Computers and Electronics in Agriculture*, vol. 31, no. 2, pp. 151–168, Apr. 2001, doi: 10.1016/S0168-1699(00)00181-2.
- [7] M. K. A. Hamdan, D. T. Rover, M. J. Darr, and J. Just, “Mass Estimation from Images using Deep Neural Network and Sparse Ground Truth,” *arXiv:1908.04387 [cs, stat]*, Sep. 2019, Accessed: Sep. 01, 2021. [Online]. Available: <http://arxiv.org/abs/1908.04387>
- [8] C. Bhatt, A. Hassanien, N. A. Shah, and J. Thik, “Barqi breed Sheep Weight Estimation based on Neural Network with Regression,” p. 8.
- [9] Chaithanya C. and Priya S., “OBJECT WEIGHT ESTIMATION FROM 2D IMAGES,” *ARPJ Journal of Engineering and Applied Sciences*, vol. 10, pp. 7574–7578, Sep. 2015.
- [10] M. Jiang and G. Guo, “Body Weight Analysis From Human Body Images,” *IEEE Trans. Inform. Forensic Secur.*, vol. 14, no. 10, pp. 2676–2688, Oct. 2019, doi: 10.1109/TIFS.2019.2904840.
- [11] M. Ester, H.-P. Kriegel, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” p. 6.

APPENDIX

Jika Anda punya lampiran dari project, silahkan dilampirkan di bagian ini. Yang wajib Anda lampirkan adalah kode program (coding) lengkap dan diberikan keterangan terlebih dahulu pada bagian atas dari coding tersebut, koding ditulis dengan format font yang berbeda. Contoh:

CONNECT TO DRIVE

```
220 !pwd
221
222 from google.colab import drive
223 drive.mount("/content/ProjectDataset/")
224
225 import os
226 os.chdir("/content/ProjectDataset/")
227 !pwd
```

COPY INPUT IMAGE TOP VIEW FROM DRIVE TO TMP FOLDER

```
228 import shutil
229 original=
    r'/content/ProjectDataset/MyDrive/ProjectDataset/InputImage/TestImage/Test2/top/lemon/lemon4_top1
    9.jpeg'
230 target = r'/tmp/inputtopview.jpg'
231
232 shutil.copyfile(original, target)
```

COPY INPUT IMAGE SIDE VIEW FROM DRIVE TO TMP FOLDER

```
233 import shutil
234
235 original=
    r'/content/ProjectDataset/MyDrive/ProjectDataset/InputImage/TestImage/Test2/side/lemon/lemon4_side
    19.jpeg'
236 target = r'/tmp/inputsideview.jpg'
237
238 shutil.copyfile(original, target)
```

REMOVE TOP VIEW INPUT IMAGE BACKGROUND TO TRANSPARENT

```
239 import numpy as np
240 import cv2 as cv
241 from matplotlib import pyplot as plt
242 import sys
243 from PIL import Image
244
245 img = cv.imread('/tmp/inputtopview.jpg', cv.IMREAD_UNCHANGED)
246 img = cv.rotate(img, cv.cv2.ROTATE_90_CLOCKWISE)
247 imagecopy = img.copy()
248
249 kernele = np.ones((8, 8), 'uint8')
```

```

250 kerneld = np.ones((4, 4), 'uint8')
251
252 filter = cv.cvtColor(img, cv.COLOR_BGR2HSV)
253 filter = cv.cvtColor(filter, cv.COLOR_HSV2RGB)
254 filter = cv.cvtColor(filter, cv.COLOR_RGB2GRAY)
255 filter = cv.GaussianBlur(filter, (17, 17), 17)
256 filter = cv.erode(filter, kernele, iterations=5)
257 filter = cv.Canny(filter, 23, 23)
258 filter = cv.dilate(filter, kerneld, iterations=5)
259
260 _, thresh = cv.threshold(filter, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
261 kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
262 mask = cv.morphologyEx(thresh, cv.MORPH_CLOSE, kernel, iterations=4)
263
264 data = mask.tolist()
265 sys.setrecursionlimit(10**8)
266 for i in range(len(data)):
267     for j in range(len(data[i])):
268         if data[i][j] != 255:
269             data[i][j] = -1
270         else:
271             break
272     for j in range(len(data[i])-1, -1, -1):
273         if data[i][j] != 255:
274             data[i][j] = -1
275         else:
276             break
277 image = np.array(data)
278 image[image != -1] = 255
279 image[image == -1] = 0
280 mask = np.array(image, np.uint8)
281
282 main = cv.bitwise_and(imagecopy, imagecopy, mask=mask)
283 main[mask == 0] = 255
284 cv.imwrite('/tmp/topviewextracted.jpg', main)
285 img = Image.open('/tmp/topviewextracted.jpg')
286 img.convert("RGBA")
287 datas = img.getdata()
288
289 datas2 = []
290 for item in datas:
291     if item[0]==255 and item[1]==255 and item[2]==255:
292         datas2.append((255, 255, 255, 0))
293     else:
294         datas2.append(item)
295
296 img.putdata(datas2)
297 img.save("/tmp/topviewextractedtr.png", "PNG")
298 plt.imshow(img)

```

REMOVE SIDE VIEW INPUT IMAGE BACKGROUND TO TRANSPARENT

```

299 import numpy as np
300 import cv2 as cv
301 from matplotlib import pyplot as plt
302 import sys
303 from PIL import Image
304

```

```

305 img = cv.imread('/tmp/inputsideview.jpg', cv.IMREAD_UNCHANGED)
306 img = cv.rotate(img, cv.cv2.ROTATE_90_CLOCKWISE)
307 imagecopy = img.copy()
308
309 kernele = np.ones((7, 7), 'uint8')
310 kerneld = np.ones((3, 3), 'uint8')
311
312 filter = cv.cvtColor(img, cv.COLOR_BGR2HSV)
313 filter = cv.cvtColor(filter, cv.COLOR_HSV2RGB)
314 filter = cv.cvtColor(filter, cv.COLOR_RGB2GRAY)
315 filter = cv.GaussianBlur(filter, (15, 15), 15)
316 filter = cv.erode(filter, kernele, iterations=2)
317 filter = cv.Canny(filter, 25, 25)
318 filter = cv.dilate(filter, kerneld, iterations=2)
319
320 _, thresh = cv.threshold(filter, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
321 kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
322 mask = cv.morphologyEx(thresh, cv.MORPH_CLOSE, kernel, iterations=4)
323
324 data = mask.tolist()
325 sys.setrecursionlimit(10**8)
326 for i in range(len(data)):
327     for j in range(len(data[i])):
328         if data[i][j] != 255:
329             data[i][j] = -1
330         else:
331             break
332     for j in range(len(data[i])-1, -1, -1):
333         if data[i][j] != 255:
334             data[i][j] = -1
335         else:
336             break
337 image = np.array(data)
338 image[image != -1] = 255
339 image[image == -1] = 0
340 mask = np.array(image, np.uint8)
341
342 main = cv.bitwise_and(imagecopy, imagecopy, mask=mask)
343 main[mask == 0] = 255
344 cv.imwrite('/tmp/sideviewextracted.jpg', main)
345 img = Image.open('/tmp/sideviewextracted.jpg')
346 img.convert("RGBA")
347 datas = img.getdata()
348
349 datas2 = []
350 for item in datas:
351     if item[0]==255 and item[1]==255 and item[2]==255:
352         datas2.append((255, 255, 255, 0))
353     else:
354         datas2.append(item)
355
356 img.putdata(datas2)
357 img.save("/tmp/sideviewextractedtr.png", "PNG")
358 plt.imshow(img)

```

EXTRACT DATA TO TMP FOLDER

```

359 import zipfile

```



```

360 original_zip = '/content/ProjectDataset/MyDrive/ProjectDataset/Training10.zip'
361 zip_ref = zipfile.ZipFile(original_zip, 'r')
362 zip_ref.extractall('/tmp/Training')
363 zip_ref.close()

```

COPY TEST IMAGE SIDE VIEW FROM DRIVE TO TMP FOLDER

```

364 import shutil
365
366 local = r'/content/ProjectDataset/MyDrive/ProjectDataset/Input Image/Test1/Test10/lemon.jpg'
367 os.mkdir('/tmp/Training/Input Image')
368 target = r'/tmp/Training/Input Image/inputimage.jpg'
369
370 shutil.copyfile(local, target)

```

STORE EXTRACTED RESULT IN TUPLE FORM

```

371 from pathlib import Path
372 import os
373 from collections import namedtuple
374 import pandas as pd
375
376 #all dataset
377 Files = namedtuple('File', 'name path')
378 dataset = []
379 p = Path('/tmp/Training/')
380 for item in p.glob('*/*/*'):
381     name = item.name
382     path = Path.resolve(item).parent
383     dataset.append(Files(name, path))
384
385 dataset

```

REMOVE UNNECESSARY ROWS

```

386 import pandas as pd
387 all_datafiles = pd.DataFrame(dataset)
388 all_datafiles
389
390 all_files = all_datafiles.iloc[11:]
391 all_files

```

IMAGE HASH CALCULATION FUNCTION

```

392 !pip install imagehash
393 import imagehash
394 import numpy as np
395 from PIL import Image
396 from itertools import combinations
397 from sklearn.cluster import DBSCAN
398 from collections import defaultdict
399
400
401 def hashes_calculation(files, hashfunc=imagehash.whash):
402     hashes, names = [], []
403     for i, name in enumerate(files):

```

```

404     try:
405         img = Image.open(name)
406         hash = hashfunc(img)
407         hashes.append(hash)
408         names.append(name)
409     except:
410         pass
411
412     return hashes, names

```

MATRIX DISTANCE CALCULATION FUNCTION

```

413 def distances_calculation(hashes):
414     matrix = np.zeros((len(hashes), len(hashes)))
415     for i, j in combinations(range(len(hashes)), 2):
416         dist = hashes[i] - hashes[j]
417         matrix[i, j] = matrix[j, i] = dist
418     return matrix

```

CALCULATE HASH FOR EACH IMAGE IN DATASET

```

419 from glob import glob
420 path = "/tmp/Training/**/*"
421 hashes, names = hashes_calculation(glob(path))

```

CALCULATE DISTANCE MATRIX FOR EACH HASHED IMAGE

```

422 matrix = distances_calculation(hashes)
423 matrix
424
425 import pandas as pd
426 df = pd.DataFrame(matrix)
427 df

```

DISTANCE MATRIX TO COORDINATE TRANSFORMATION

```

428 import numpy as np
429 from sklearn.decomposition import PCA
430 pca = PCA(n_components=2)
431 dist2d = pca.fit_transform(matrix)
432 dist2d
433
434 import pandas as pd
435 df = pd.DataFrame(dist2d)
436 df.to_csv('/content/ProjectDataset/MyDrive/coordinates.csv', index=False)

```

CREATE DBSCAN CLASS

```

437 import numpy as np
438 from sklearn import datasets
439 from sklearn.preprocessing import StandardScaler
440 from itertools import cycle, islice
441 import matplotlib.pyplot as plt
442 import queue
443 import pandas as pd
444

```

```

445
446 class DBSCAN():
447     def __init__(self):
448         self.core = -1
449         self.border = -2

```

DBSCAN FIND NEIGHBOUR FUNCTION

```

450     def find_neighbour(self, data, point_id, eps):
451         points = []
452         for i in range(len(data)):
453             # Euclidian distance
454             if np.linalg.norm([a_i - b_i for a_i, b_i in zip(data[i], data[point_id])]) <= eps:
455                 points.append(i)
456         return points

```

DBSCAN FIT FUNCTION

```

457     def fit(self, data, Eps, MinPts):
458         point_label = [0] * len(data)
459         point_count = []
460
461         core = []
462         border = []
463
464         for i in range(len(data)):
465             point_count.append(self.find_neighbour(data, i, Eps))
466
467         for i in range(len(point_count)):
468             if (len(point_count[i]) >= MinPts):
469                 point_label[i] = self.core
470                 core.append(i)
471             else:
472                 border.append(i)
473
474         for i in border:
475             for j in point_count[i]:
476                 if j in core:
477                     point_label[j] = self.border
478                     break
479
480         cluster = 1
481
482         for i in range(len(point_label)):
483             q = queue.Queue()
484             if (point_label[i] == self.core):
485                 point_label[i] = cluster
486                 for x in point_count[i]:
487                     if (point_label[x] == self.core):
488                         q.put(x)
489                         point_label[x] = cluster
490                     elif (point_label[x] == self.border):
491                         point_label[x] = cluster
492             while not q.empty():
493                 neighbors = point_count[q.get()]
494                 for y in neighbors:
495                     if (point_label[y] == self.core):

```

```

496         point_label[y] = cluster
497         q.put(y)
498         if (point_label[y] == self.border):
499             point_label[y] = cluster
500         cluster += 1 # Move on to the next cluster
501
502     return point_label, cluster

```

DBSCAN VISUALIZATION FUNCTION

```

503     def visualize(self, data, cluster, clusters_count):
504         N = len(data)
505
506         colors = np.array(list(islice(cycle(['#FE4A49', '#2AB7CA']), 3)))
507
508         for i in range(clusters_count):
509             if (i == 0):
510                 color = '#000000'
511             else:
512                 color = colors[i % len(colors)]
513
514             x, y = [], []
515             for j in range(N):
516                 if cluster[j] == i:
517                     x.append(data[j, 0])
518                     y.append(data[j, 1])
519             plt.scatter(x, y, c=color, alpha=1, marker='.')
520         plt.show()

```

DBSCAN CLUSTERING

```

521 df = pd.read_csv("/content/ProjectDataset/MyDrive/coordinates.csv")
522 dataset = df.astype(float).values.tolist()
523 X = StandardScaler().fit_transform(dataset)
524 DBSCAN = DBSCAN()
525 point_labels, clusters = DBSCAN.fit(X, 0.1, 2)
526 print(point_labels, clusters)
527 DBSCAN.visualize(X, point_labels, clusters)
528 print(clusters)
529 len(point_labels)

```

SELECT INPUT IMAGE CLUSTER ONLY

```

530 import pandas as pd
531 df = pd.DataFrame(point_labels, columns=['Point Labels'])
532 df
533
534 all_files.insert(2, "Point Labels", point_labels)
535 all_files
536
537 input_img = all_files[all_files['name'] == "inputimage.jpg"]
538
539 pntlbl = int(input_img['Point Labels'])
540 filter_data = all_files[all_files['Point Labels'] == pntlbl]
541 filter_data

```

NEW DATA FOR K-NN CONSIST OF IMAGES WITH THE SAME CLUSTER AS INPUT IMAGE

```
542 new_data = filter_data
543 new_data
544
545 new_data["filepath"] = new_data["path"] / new_data["name"]
546 new_data
547
548 new_data.reset_index(drop=True, inplace=True)
549 new_data
```

SWITCH THE FIRST ROW WITH INPUT IMAGE FOR K-NN REFERENCE POINT

```
550 switch = new_data.index[new_data['name'] == "inputimage.jpg"].tolist()
551 switch = switch[0]
552 switch
553
554 b = new_data.iloc[switch]
555 temp = new_data.iloc[0].copy()
556 new_data.iloc[0] = b
557 new_data.iloc[switch] = temp
558
559 new_data
```

IMAGE HASH CALCULATION

```
560 a = []
561 for i, row in new_data.iterrows():
562     knnpattern = str(row['filepath'])
563     hashes, frames = hashes_calculation(glob(knnpattern))
564     b = list(hashes)
565     a = a + b
```

DISTANCE MATRIX CALCULATION

```
566 matrix2 = distances_calculation(a)
567
568 import pandas as pd
569 df = pd.DataFrame(matrix2)
570 df
```

DISTANCE MATRIX TO COORDINATE TRANSFORMATION

```
571 import numpy as np
572 from sklearn.decomposition import PCA
573 pca = PCA(n_components=2)
574 dist2d2 = pca.fit_transform(matrix2)
575
576 dist2d2
```

LABELLING IMAGES COORDINATES

```
577 import pandas as pd
578 df = pd.DataFrame(dist2d2)
```

```

579 df.columns =['X', 'Y']
580 df
581
582 result = pd.concat([df, new_data.reindex(df.index)], axis=1)
583 result

```

K-NN EUCLIDEAN DISTANCE FUNCTION

```

584 from math import sqrt
585
586 # calculate the Euclidean distance between two vectors
587 def euclidean_distance(vec1, vec2):
588     distance = 0.0
589     for i in range(len(vec1)-1):
590         distance += (vec1[i] - vec2[i])**2
591     return sqrt(distance)

```

K-NN GET NEIGHBOURS FUNCTION

```

592 def get_neighbors(train, test_row, num_neighbors):
593     distances = list()
594     for train_row in train:
595         dist = euclidean_distance(test_row, train_row)
596         distances.append((train_row, dist))
597     distances.sort(key=lambda tup: tup[1])
598     neighbors = list()
599     for i in range(num_neighbors):
600         neighbors.append(distances[i][0])
601     return neighbors

```

K-NN CLASSIFICATION

```

602 n = []
603 dataset = dist2d2
604 neighbors = list(get_neighbors(dataset, dataset[0], 2))
605 n = n + neighbors
606 print (n)

```

DESCRIBE THE RESULT OF THE NEAREST NEIGHBOUR

```

607 import pandas as pd
608 ans = pd.DataFrame(n)
609 ans.columns =['X', 'Y']
610 array = ans.to_numpy()
611 array
612
613 import numpy as np
614 res = array[1]
615 newres = np.array_split(res, 2)
616 newres
617 finalx = float(newres[0])
618 finalx
619
620 fin = result[np.isclose(result['X'],finalx, 0.0000000001)]
621 fin = fin.iloc[1: , :]
622 fin

```

EXTRACT THE OBJECT NAME

```
623 n = fin.iloc[:1]
624 n = n['path']
625 txt = n.to_string(index=False)
626 r = txt.split("/", 3)
627 fr = r[3]
628 fr
```

GET OBJECT DENSITY

```
629 import pandas as pd
630 dens = pd.read_csv("/content/ProjectDataset/MyDrive/ProjectDataset/density/Fruit_density.csv", sep=',')
631 dens
632
633 final_dense = dens[dens['Fruit'] == fr]
634 final_dense
635
636 density = float(final_dense['Density (g/cm3)'])
637 density
```

REPLACE TOP VIEW IMAGE TRANSPARENT BACKGROUND WITH BLACK

```
638 import sys
639 import numpy as np
640 import skimage.color
641 import skimage.filters
642 import skimage.io
643
644 image = skimage.io.imread(fname='/tmp/topviewextractedtr.png')
645 blur = skimage.color.rgb2gray(image)
646 mask = blur < 0.98
647 blackbg = np.zeros_like(image)
648 blackbg[mask] = image[mask]
649 skimage.io.imshow(blackbg)
```

OBJECT'S AREA CALCULATION

```
650 import cv2
651 import numpy as np
652 img = blackbg
653 height = img.shape[0]
654 width = img.shape[1]
655 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
656 ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)
657 plt.imshow(thresh)
658 count = cv2.countNonZero(thresh)
659 area = count*0.14*0.14/(width*height)
660 print(area)
```

REPLACE SIDE VIEW IMAGE TRANSPARENT BACKGROUND WITH BLACK

```
661 import sys
662 import numpy as np
663 import skimage.color
664 import skimage.filters
```

```

665 import skimage.io
666
667 image = skimage.io.imread(fname='/tmp/sideviewextractedtr.png')
668 skimage.io.imshow(image)
669 blur = skimage.color.rgb2gray(image)
670 mask = blur < 0.98
671 blackbg2 = np.zeros_like(image)
672 blackbg2[mask] = image[mask]
673
674 skimage.io.imshow(blackbg2)

```

OBJECT'S HEIGHT ESTIMATION

```

675 import cv2
676 import numpy as np
677 img = blackbg2
678 gray2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
679 ret, thresh2 = cv2.threshold(gray2,0,255,cv2.THRESH_BINARY)
680 plt.imshow(thresh2)
681
682 image2 = skimage.io.imread(fname="/tmp/sideviewextractedtr.png")
683 x,y,w,h = cv2.boundingRect(thresh2)
684 cv2.rectangle(image2, (x, y), (x + w, y + h), (36,255,12), 2)
685 cv2.putText(image2, "w={},h={}".format(w,h), (x,y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
        (36,255,12), 2)
686 plt.imshow(image2)
687 print(h)

```

OBJECT'S VOLUME CALCULATION

```

688 V = round(area*10000)*round((h/100)/2)
689 V

```

OBJECT'S MASS CALCULATION

```

690 M = density * V
691 M
692
693 round(M)

```


Bonita Nugroho Widjanarko

18.K1.0001.docx

Sources Overview

9%

OVERALL SIMILARITY

1	arxiv.org	INTERNET	2%
2	ntnuopen.ntnu.no	INTERNET	1%
3	www.di.unipi.it	INTERNET	1%
4	www.arpnjournals.com	INTERNET	<1%
5	Min Jiang, Guodong Guo. "Body Weight Analysis From Human Body Images", IEEE Transactions on Information Forensics and Securit...	CROSSREF	<1%
6	ebin.pub	INTERNET	<1%
7	www.silsoeresearch.org.uk	INTERNET	<1%
8	proceedings.mlr.press	INTERNET	<1%
9	University of Computer Studies on 2016-08-12	SUBMITTED WORKS	<1%
10	K. Kollis, , C. S. Phang, T. M. Banhazi, and S. J. Searle. "Weight Estimation Using Image Analysis and Statistical Modelling: A Preliminar...	CROSSREF	<1%
11	Nirav Alpesh Shah, Jaydeep Thik, Chintan Bhatt, Aboul-Ella Hassanien. "Chapter 4 A Deep Convolutional Encoder-Decoder Architectur...	CROSSREF	<1%
12	ieeexplore.ieee.org	INTERNET	<1%
13	Radhamadhab Dalai, Kishore Kumar Senapati. "A Heuristic Grid Area Based Segmentation Approach for Weight Estimation of an Obje...	CROSSREF	<1%
14	Guihong Fu, Yun Yuna. "Phenotyping and phenomics in aquaculture breeding", Aquaculture and Fisheries, 2021	CROSSREF	<1%
15	KDU College Sdn Bhd on 2019-12-05	SUBMITTED WORKS	<1%

Excluded search repositories:

None

Excluded from document: