SUMMER INTERNSHIP

IMAGE ANALYSIS APPLICATION TO DETERMINE GRADING OF PAVEMENT MIXTURES

Submitted in partial fulfillment of the
requirement for the award of the degree
of

BACHELOR OF TECHNOLOGY

in

CIVIL ENGINEERING

By

PRANAV BARADKAR
Roll No.: 17CE02012

With the Guidance
Of

Dr. Anush K. Chandrappa

SCHOOL OF INFRASTRUCTURE
INDIAN INSTITUTE OF TECHNOLOGY BHUBANESWAR
ARGUL, JATNI -752050, ODISHA

30 JUNE 2020

CONTENTS

List of Figures

CHAPTER-1

INTRODUCTION

1. General

The pavements can be classified based on the structural performance into two as flexible pavements and rigid pavements. Inflexible pavements, wheel loads are transferred by grain-to-grain contact of the aggregate through the granular structure. The flexible pavement, having less flexural strength, acts like a flexible sheet (e.g. bituminous road). On the contrary, in rigid pavements, wheel loads are transferred to sub-grade soil by flexural strength of the pavement and the pavement acts as a rigid plate (e.g. cement concrete roads). In addition to these, composite pavements are also available. A thin layer of flexible pavement over rigid pavement is an ideal pavement with the most desirable characteristics.

Patterned layers of a conventional flexible pavement include seal coat, surface course, tack coat, binder course, prime coat, base course, sub-base course, compacted sub-grade, and natural sub-grade. Seal coat is a thin surface treatment used to water-proof the surface and to provide skid resistance. Tack coat is a very light application of asphalt, usually asphalt emulsion diluted with water. It gives proper bonding between two layers of binder course and must be thin, uniformly cover the entire surface, and set very fast. A prime coat is an application of low viscous cutback bitumen to an absorbent surface like granular bases on which binder layer is placed. It provides bonding between two layers. Unlike tack coat, prime coat penetrates the layer below, plugs the voids, and forms a watertight surface. Surface course is the layer directly in contact with traffic loads and generally contains superior quality materials. They are usually constructed with dense graded asphalt concrete (AC). Binding Course layer provides the bulk of the asphalt concrete structure. The base course is the layer of material immediately beneath the surface of the binder course, and it provides additional load distribution and contributes to the sub-surface drainage. It may be composed of crushed stone, crushed slag, and other untreated or stabilized materials. The sub-base course is the layer of material beneath the base course, and the primary functions are to provide structural support, improve drainage, and reduce the intrusion of fines from the sub-grade in the pavement structure If the base course is open-graded, then the sub-base course with more fines can serve as a filler between sub-grade and the base course A sub-base course is not always needed or used. The topsoil or sub-grade is a layer of natural soil prepared to receive the stresses from the layers above.

Based on the nature of gradation selected for the bitumen mixes, they can be classified into 4 types, as shown in Figure 1. :

- Dense Graded Bitumen Mixes
- Semi-Dense Graded Bitumen Mixes
- Open Graded Bitumen Mixes
- Gap Graded Bitumen Mixes

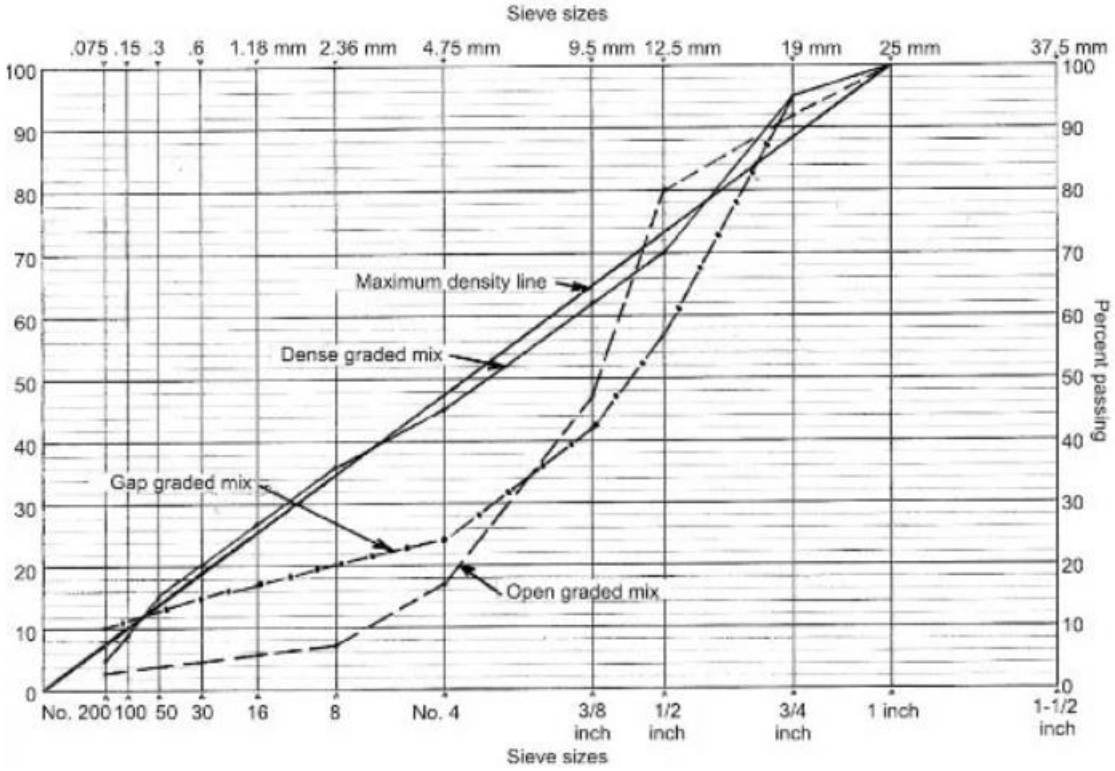

Figure 1: Grading Curve

The bitumen mix that is densely graded has continuous gradation, say in the proximity of the maximum density line. The bitumen mix with a large amount of fine aggregate, i.e. sand, will form open-graded bitumen mix. When the mix lack materials of two or more sizes, it will form the gap graded bitumen mix. The semi-graded mix will have a gradation lying in between the open-graded and the gap graded.

2. Problem Statement:

The aggregate gradation is one of the most important parameters in the mechanical properties of conventional flexible pavement. As part of quality control and assurance, all the highway projects involving bituminous mixtures require cores to be taken on a daily basis to determine gradation and bitumen content added in the mixture. To determine gradation and bitumen content, centrifuge extraction method is used, where bitumen solvents such as trichloroethylene (TCL) is used. It has been found that TCL is carcinogenic, which has the potential to cause cancer affecting the health of labours as it is used in most of the laboratories. Further, the continuous inhalation of vapours from TCL will cause irritation in the respiratory system. However, determining the gradation and bitumen content is an important quality control test and therefore, cannot be neglected. In this regard, innovative methods can be conceptualized to determine the gradation of bituminous mixtures using methods such as image processing and Analysis. In this study, a simple approach is introduced to quickly and easily determine the aggregate gradation of HMA from the prepared cross-section images of cylindrical samples using numerical and image-processing techniques such as fitting equation and connected component method. The obtained results indicate that the introduced method can detect the aggregate gradation with high accuracy and can be used as a satisfactory alternative to other expensive methods.

3. Objective of the Study

The main objective of the study was to develop a framework to determine the gradation of the aggregate used in the bituminous mixtures.

The scope of the study involved

- Understanding images
- Image processing and analysis techniques
- Application of image processing and Analysis for estimating gradation

In this study only preliminary Analysis was made as images for Analysis were obtained randomly from Transportation Engineering Laboratory, IIT Bhubaneswar.

CHAPTER-2

IMAGE PROCESSING & ANALYSIS

1. Introduction

Image processing and Analysis is a branch of electrical engineering, which has wide applications in many fields of engineering. In the field of pavement engineering, image processing and Analysis has wide range of applications. Image processing and Analysis can be used for identifying cracks and rutting, identifying the texture of the pavement, determine the gradation of the mixture, determine the contact points and orientation of the aggregates, and also as a surrogate measure for mechanical tests. This report primarily deals with processing and Analysis of image for studying the grading curve of aggregate and determining various mixture parameters through extracted data.

2. Image

Image is a collection of element called pixels. Pixels are picture elements, where each pixel is assigned a particular value of intensity. When all the pixels are arranged together, an image is obtained. When image is clicked by the camera, intensity value is assigned for each pixel in the form of value of Red-Green-Blue (RGB). Every colour (RGB) can have pixels with intensity ranging from 0 to 255, which means each colour can have 256-pixel intensities. Therefore, a colour image can be considered as a stack of red-green-blue coloured pixels, where the total number of colour combinations can be 16777216 colours. However, the RGB images are computationally expensive to analyze and a simple approach is to convert RGB image to a Gray scale image. The Gray scale image consists of 256 pixels with intensities ranging from 0 to 255, where ‘0’ indicates black and ‘255’ indicates complete white. Any other pixel intensity will be having Gray colour of varying tone. The Gray scale images are simple to process and analyze and hence most of the techniques require the images to be converted to grayscale before processing. Further, the Gray scale images are later converted to binary images, which will be used for the Analysis. The image processing and Analysis typically involves the following steps:

- Image acquisition
- Point processing
- Neighbourhood processing
- Image restoration

- Image segmentation
- Image analysis

Image acquisition deals with the methods involved in acquiring images. The methods used may depend on the type of camera, lighting, contrast, etc.

3. Image processing

The images which are acquired may be having some unwanted features such as lack of contrast between foreground and background, unwanted noise such as salt & pepper noise, etc. which needs to be corrected to obtain clear information from the image. Further, the images have be processed to segment the image, which forms the input for the image analysis. The image processing uses point processing and neighbourhood processing techniques to apply corrective measures for the images. The images are further thresholded or segmented to convert them into binary image for image analysis using methods such as connected component analysis.

4. Image analysis

The binary image from the image processing forms the primary input for the image analysis. The most commonly used methods to analyze binary image is connected component analysis which will give various details of the objects and their dimensions.

CHAPTER-3

METHODOLOGY

1. General

In this study the morphological properties of the mixtures, 2D planar images were utilized. 2D planar images can be obtained by sectioning the asphalt mixture either vertically or horizontal and scanning using a flatbed scanner.

Before image using for Analysis some manipulation is needed (pre-processing) to get the most accurate result. Pre-processing can be done as per the following method.

• Image acquisition

• Point processing

• Neighbourhood processing

• Image restoration

• Image segmentation

After pre-processing of the image, we will get needful images for each sample. Connected component method will help in extracting areal data from the image such as area of cement paste, area of each aggregate and are of air voids.

For the pre-processing and Analysis python language and open-cv framework was used in entire report.

2. Methodology

For Pervious Concrete Specimens:

```
img = cv.imread (cv.samples.findFile("G:\my Ml projects\image processing\p5.PNG"))
```

The above code snippet takes the image from the local path and convert it into a NumPy array based of RGB values of each pixel.

For converting RGB image into Grayscale; following code is used.

```
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
```

If we see mathematically, it add pixel value of R, G, and B of every pixel and take an average of it and later convert the image into grayscale image with pixel ranging from 0-255.

Grayscale image needs to be further processed to obtain reliable and accurate results. One of the most important requirements for image analysis is the contrast between the foreground and the background should be more.

- The contrast in the image can be enhanced for better visualization
- Histogram of the grayscale image is an indicator of the contrast
- Histogram which is skewed towards left end or right end indicates a poor contrast
- An image with good contrast has a broad histogram with pixels at all the intensity levels

Histogram equalization method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram and hence the image. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values as shown in Figure 2-6.

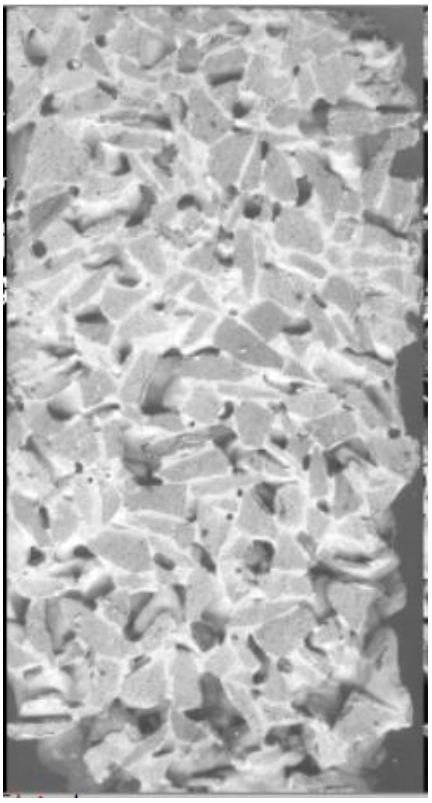

Figure 2: Grayscale image

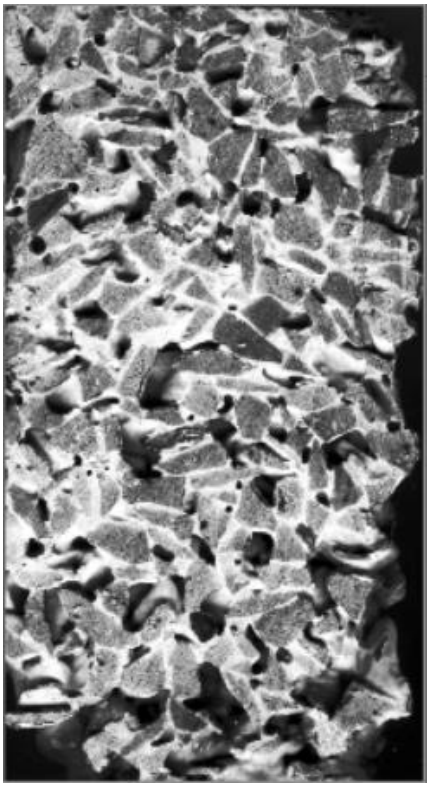

Figure 3: Grayscale image after histogram equalization

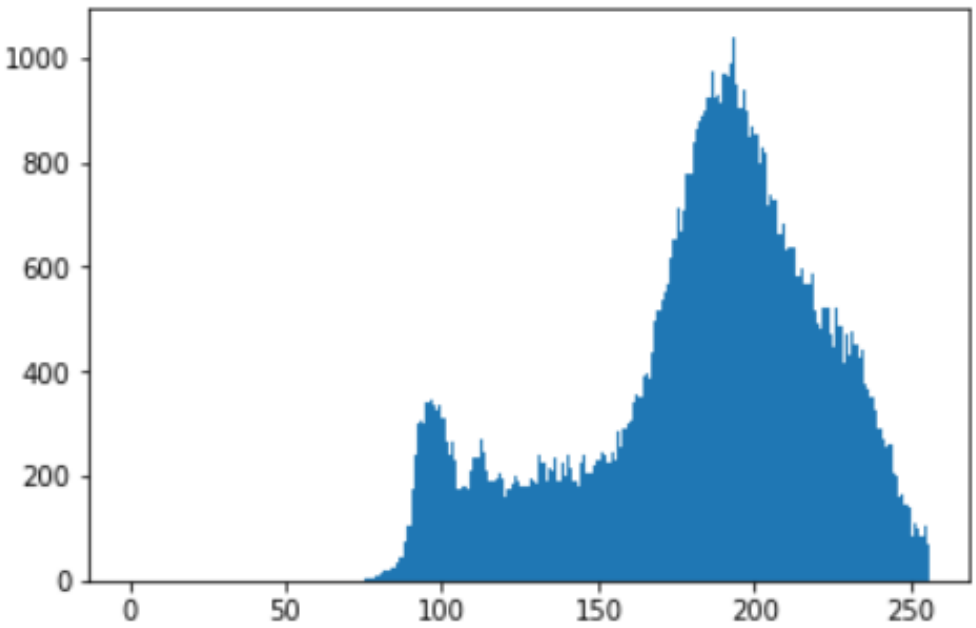

Figure 4: Histogram of Original Gray Scale Image

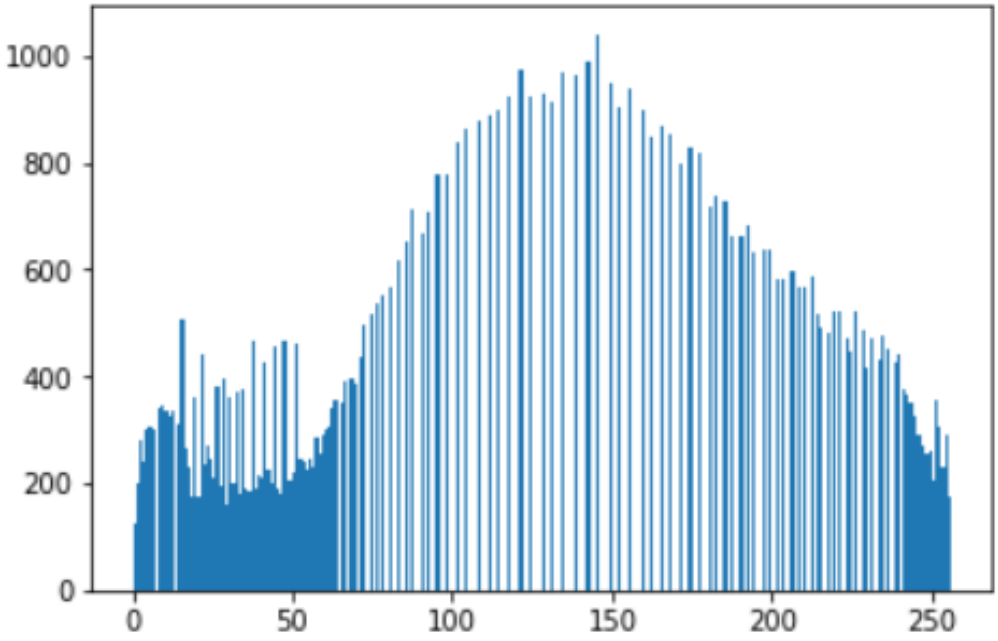

Figure 5: Automatic equalize histogram (Normalization)

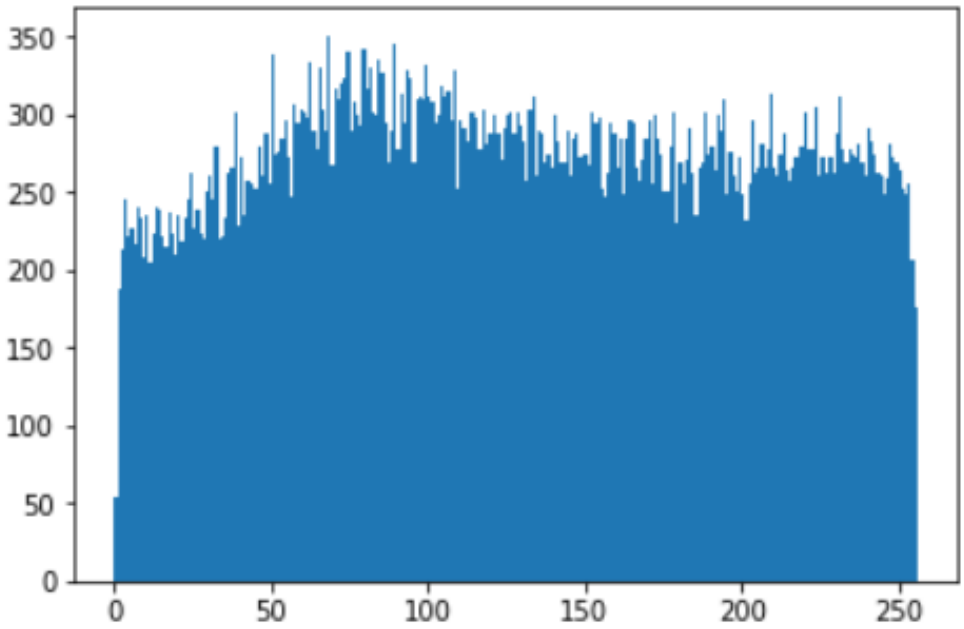

Figure 6: Contrast limited adaptive histogram equalization (CLAHE)

Normalization: Refer Figure - 4

equ1 = cv.equalizeHist(gray)

This python code stretches the histogram to equalize the pixel values over the range. As shown in Figure 4. normalization is a process that changes the range of pixel intensity values.

Applications include photographs with poor contrast due to glare, for example. Normalization is sometimes called contrast stretching or histogram stretching.

Normalization transforms an n-dimensional grayscale image with intensity values in the range (Min,Max), into a new image with intensity values in the range (newMin,newMax).

New value of pixel intensity will be:

$$I_n = (I - Min)\ \frac{newMax - newMin}{Max - Min} + (newMin) \qquad \text{Eq.1}$$

I_n = New intensity value of pixel

I = Original pixel intensity value

Contrast Limited Adaptive Histogram Equalization: As shown in Figure 5

clahe = cv.createCLAHE(clipLimit=12.0, tileGridSize=(8,8))

equ = clahe.apply(gray)

Image is divided into small blocks called "tiles" (tile Size is 8x8 by default in OpenCV). Then each of these blocks are histogram equalized as usual. Therefore, in a small area, histogram would confine to a small region (unless there is noise). If noise is there, it will be amplified. To avoid this, contrast limiting is applied. If any histogram bin is above the specified contrast limit (by default 40 in OpenCV), those pixels are clipped and distributed uniformly to other bins before applying histogram equalization. After equalization, to remove artifacts in tile borders, bilinear interpolation is applied.

- ***Filter:***

Some images have extra noise which can make errors in Analysis when we convert image in binary image. Filters helps to reduce that noise.

In neighbourhood processing, a function is applied to a pixel and its neighbourhood. The basic concept is to move a mask (1D or 2D) over the image and a mathematical operation called “convolution” will be carried out, which will alter the pixel and obtain a new image.

The combination of a mask and function is called a filter. The 1D or 2D mask moves over the image pixels. Multiply each value under the mask matrix with pixel values add it and then give the new grayscale value to the pixel.

blur = cv.bilateralFilter(equ,5,51,51)

cv.bilateralFilter() is highly effective in noise removal while keeping edges sharp. However, the operation is slower compared to other filters. Gaussian filter takes the neighbourhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It doesn't consider whether pixels have almost the same intensity. It does not consider whether a pixel is an edge pixel or not. So, it blurs the edges also, which we do not want to do.

Bilateral filtering also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The Gaussian function of space makes sure that only nearby pixels are considered for blurring, while the Gaussian function of intensity difference makes sure that only those pixels with similar intensities to the central pixel are considered for blurring. Therefore, it preserves the edges since pixels at edges will have large intensity variation.

In pavement images we need to keep the edges sharp because it will help to make most accurate results about results. Final step for pre-processing of an image is converting well contrasted image into binary image. Binary image is an image which has only two intensity value pixels. That is, 0 for black and 255 for white. When we give some threshold then the python function will make all the intensity above threshold equals to 255 and below values to 0.

From equalized histogram, manual threshold value has been taken.

In cement paste pavement mixture, we have three components.

1. Cement paste
2. Aggregate
3. Air voids

For finding results, three binary images had taken. One is for cement paste, second is for air voids and third is for aggregate as shown in Figure 7-9.

ret,th = cv.threshold(equ,170,255,cv.THRESH_BINARY) #for cement paste

ret,th1 = cv.threshold(equ1,50,255,cv.THRESH_BINARY_INV) # for air voids

cv.threshold(equ,170,255,cv.THRESH_BINARY) converts pixels whose intensity is greater than 170 to 255 and pixels whose intensity is below 170 converts to 0. Value of the threshold decided by analyzing equalized histogram and result we got when manually changes the value.

cv.threshold(equ1,50,255,cv.THRESH_BINARY_INV)

cv.THRESH_BINARY_INV converts pixels whose intensity is greater than 50 to 0 and pixels whose intensity is below 50 converts to 255.

This binary inversion image is used to make a binary image of aggregate.

Figure 7: Cement paste (Represented in white)

Figure 8: Air Voids (Represented in white)

By observing above two images we can make image for aggregate by adding both the images. After addition of both the images if binary inversion happens then white pixels has given aggregate.

dst1 = cv.addWeighted(th,1,th1,1,0) #added image 1 and 2

ret,dst = cv.threshold(dst1,150,255,cv.THRESH_BINARY_INV) #for binary inversion

Figure 9: Aggregate image (Represented in white)

The pre-processed images in this way can be further used in the analyzing the images to quantify various properties.

Analysis:

Two steps have done for Analysis.

1. Total area of aggregate, cement paste and air voids.
2. Area of each aggregate using connected components method.

Total Area of aggregate, Cement paste and Air voids:

For each binary image based on the thresholding, three segmented images were obtained representing the pixels of cement paste, air voids and aggregates. In order to find the area of each component, the number of pixels belonging to each component should be first determined. The following python code provides the number of pixels belonging to each component. The images used for each component was thresholded as discussed in the previous section.

n_white_pix_w = np.sum(th2==255) #total pixels in the whole image

n_white_pix_v = np.sum(th1==255) #White pixels in air voids image

n_white_pix_c = np.sum(th==255) #White pixels in cement paste

n_white_pix_a = np.sum(dst==255) #White pixels in aggregate image

Now, the image should be calibrated in terms of geometric scale to determine the area in mm^2. The height of the specimen of pervious concrete was 200 mm and diameter was 100 mm in linear scale. Therefore the geometric area of the specimen will be 100*200 mm^2 as image in planar. The geometric scaling is performed using the following code:

Following python code do the geometric scaling to find out area.

dimensions = gray.shape

specimen_area = 100 * 200

number_of_total_pixels = dimensions[0] * dimensions[1]

mm_square_per_pixel = specimen_area / number_of_total_pixels

area_of_v = n_white_pix_v * mm_square_per_pixel #area of voids

area_of_c = n_white_pix_c * mm_square_per_pixel #area of cement

area_of_a = n_white_pix_a * mm_square_per_pixel #area of aggr.

Area of each aggregate using connected components method:

- **<u>Connected component:</u>**

  Connected components labelling scans an image and groups its pixels into components based on pixel connectivity, i.e. all pixels in a connected component share similar pixel intensity values and are in some way connected with each other. Once all groups have been determined, each pixel is labelled with a Gray level or a colour (colour labelling) according to the component it was assigned to.

  Extracting and labelling of various disjoint and connected components in an image is central to many automated image analysis applications.

- **How is it work?**

The algorithm involves two whole passes through each pixel in the image as shown in Figure 10-11.

First pass:

For each non-zero pixel, its neighbours are checked.

If it has no non-zero neighbours — it is considered as a new component and hence a new label is given.

If it has one non-zero neighbour, then these pixels are connected and hence same label is given as the neighbour.

If it has more than one non-zero neighbour, there are two cases:

Case-1: The neighbours all have the same label. In this case, the pixel get the same label to the current pixel.

Case-2: The neighbours have different labels. This is the tricky part as it is difficult to assign label as it seems like it the current pixel is connected to two different neighbours. The classical approach to solve this is the following. The current is set to the neighbour's lowest label. A note on equivalences of all the connected labels — that is, which different labels should actually be the same are kept. These equivalences are adjusted in the second pass as discussed below.

Each non-zero pixel will now have a label. However, some connected regions will have different labels. In order to assign a label, one more pass is made over the image considering the equivalences. This is done by recording the equivalence classes: all the labels that are equivalent (i.e. refer to the same blob), will get the same label.

Second pass:

For each pixel with a label:

Check if this label has equivalent labels and solve them.

It gives particular label (in the form of colour) for particular connected component(aggregate) as shown in Figure – 12.

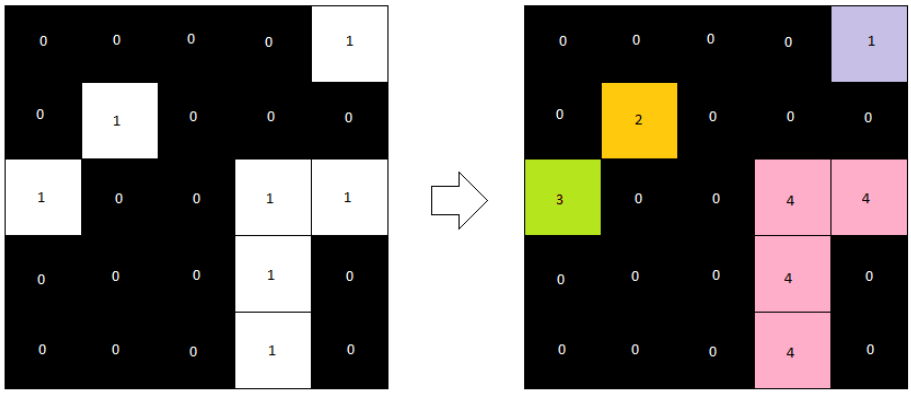

Figure 2. Result of the algorithm if you assume a **4-connectivity** representation. The '2' is not a neighbour of '3' in this case.

Figure 10: 4-connectivity representation

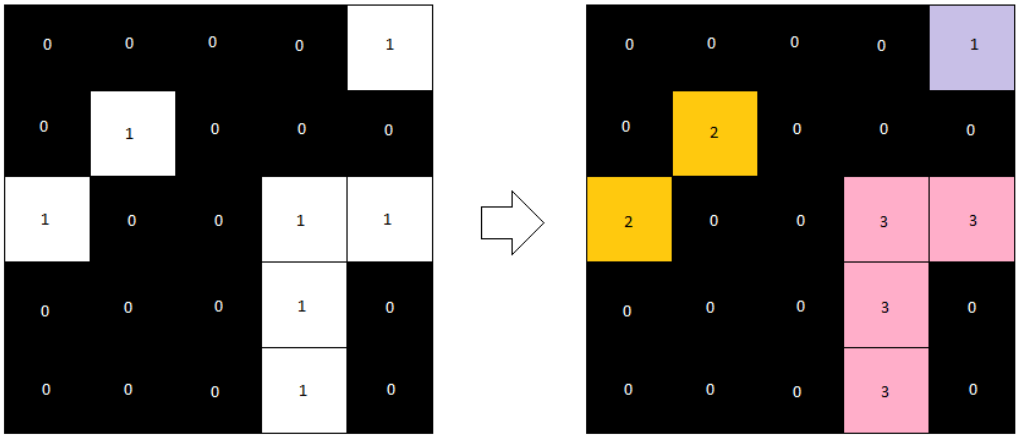

Figure 3. Result of the algorithm if you assume a **8-connectivity** representation. The "2's" are neighbours.

Figure 11: 8-connectivity representation

Pseudo code for Connected component labelling:

algorithm Two Pass(data) **is**

linked = []

labels = structure with dimensions of data, initialized with the value of Background

First pass

for row **in** data **do**

for column **in** row **do**

if data[row][column] **is not** Background **then**

```
        neighbors = connected elements with the current element's value

        if neighbors is empty then
            linked[NextLabel] = set containing NextLabel
            labels[row][column] = NextLabel
            NextLabel += 1

        else

            Find the smallest label

            L = neighbors labels
            labels[row][column] = min(L)
            for label in L do
                linked[label] = union(linked[label], L)

    Second pass

    for row in data do
        for column in row do
            if data[row][column] is not Background then
                labels[row][column] = find(labels[row][column])

    return labels
```

There is a function in opencv to deal with the above psudo code for Connecting component method.

num_labels, labels = cv.connectedComponents(dst,connectivity=8)

connectivity is an argument which takes value 4 or 8 as shown above.

Map component labels to hue val, 0-179 is the hue range in OpenCV

label_hue = np.uint8(179*labels/np.max(labels),axis=1)

blank_ch = 255*np.ones_like(label_hue)

labeled_img = cv.merge([label_hue, blank_ch, blank_ch])

This converts each label pixel into hue value colour.

labeled_img[label_hue==0] = 0

This converts black pixel label to 0.

labels = labels.reshape(np.size(gray))

print(np.shape(labels))

label_list = labels.tolist()

label_set = set(label_list)

total_labels = list(label_set)

for label in total_labels:

print("area of each label image {} = ".format(label),label_list.count(label) * mm_square_per_pixel)

This give us the area of each label. i.e. area of each aggregate.

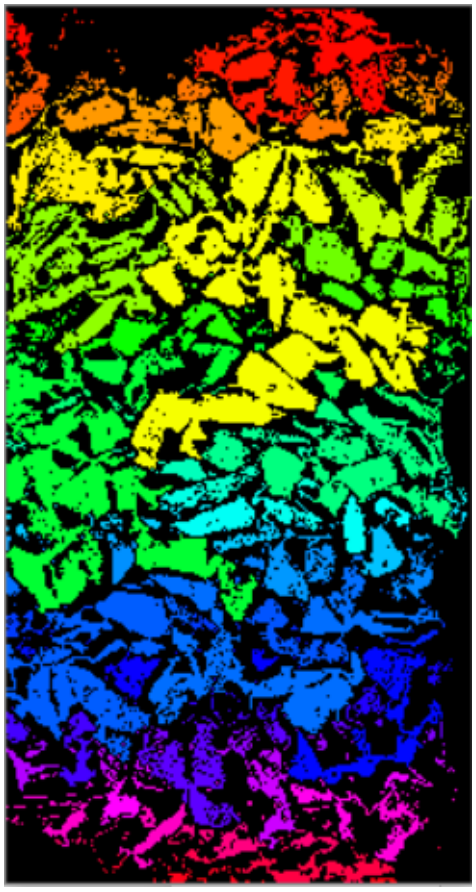

Figure 12: Aggregate classification using connected component method

For bitumen pavement:

Bitumen Pavement has only two components. As shown in Figure – 13.

1. Bitumen
2. Aggregate

gray1 = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

This will convert RGB image into grayscale as shown in Figure 14.

Figure 13: Original image

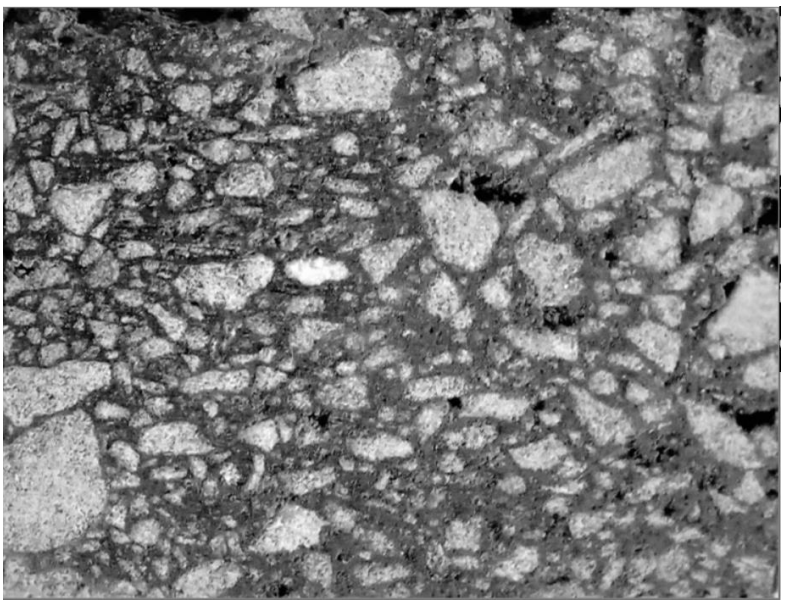

Figure 14: Grayscale Image

Bitumen pavement sample has lot of gaussian noise so using bilateral filter noise has been removed.

median = cv.bilateralFilter(img,2,75,75)

median = cv.bilateralFilter(median,2,75,75)

median = cv.bilateralFilter(median,2,75,75)

gray = cv.cvtColor(median,cv.COLOR_BGR2GRAY)

Refer Figure - 15

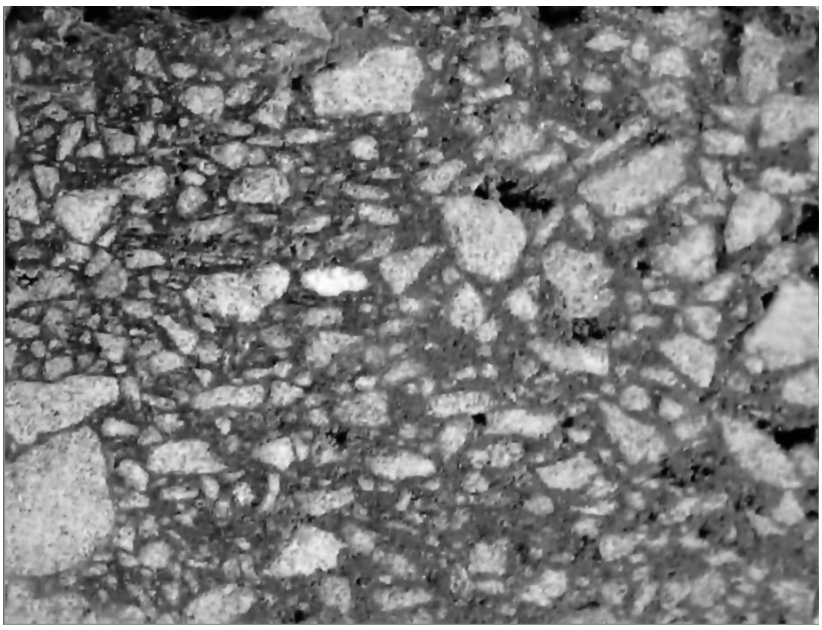

Figure15: Image after applying bilateral filters

Same like cement paste sample, histogram stretching equalization is applied after filtering the image. As shown in Figure -16.

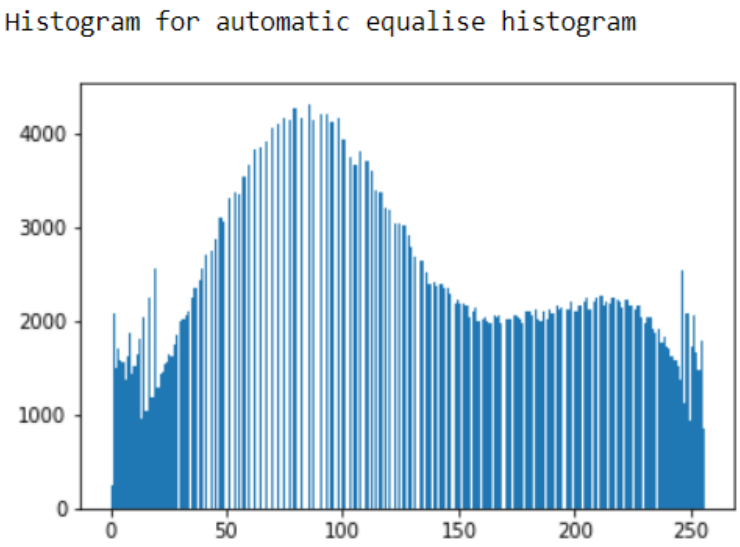

Figure16: Automatic equalize histogram (Normalization)

Before going for the analysis image has been converted to binary using following code. As shown in Figure -17

ret,th=cv.threshold(equ1,140,255,cv.THRESH_BINARY)

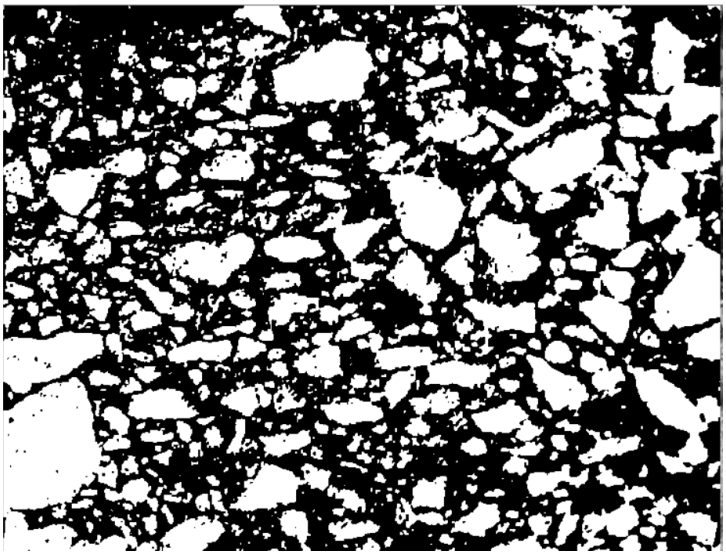

Figure17: Binary image

After pre-processing of the image. Formed image is used for Analysis.

Using connected component method area of each aggregate and using calculation of white pixel method area of whole aggregates and area of bitumen is found out. As shown in Figure 18.

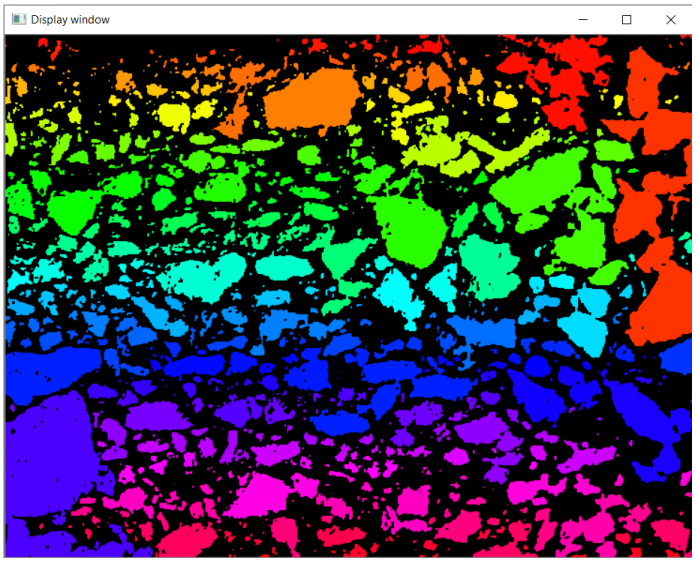

Figure18: Image after connected component analysis

3. Result and Analysis:

Gradation result of bitumen sample –

Preliminary results:

Bitumen sample (Dimensions: 7 cm by 6 cm). *Refer Figure - 19*

Figure19: *Bitumen sample (Dimensions: 7 cm by 6 cm)*

Grading Curve of Bitumen sample (Dimensions: 7 cm by 6 cm) shown in Figure – 20.

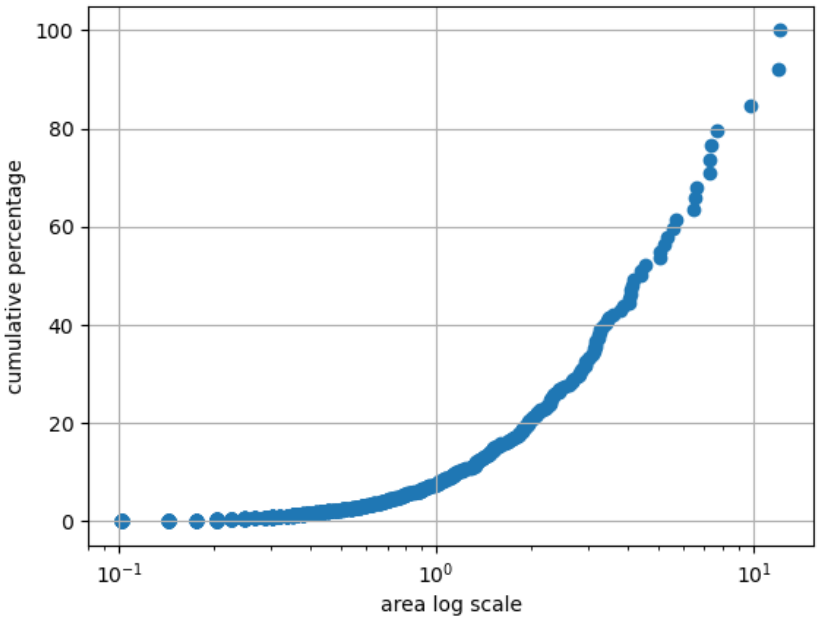

Figure 20: Analyzed grading curve for sample shown in Figure - 19

Bitumen sample (Dimensions: 30 cm by 5 cm).*Refer Figure - 21*

Figure 21: *Bitumen sample (Dimensions: 30 cm by 5 cm)*

Grading Curve of Bitumen sample (Dimensions: 30 cm by 5 cm) shown in Figure – 22.

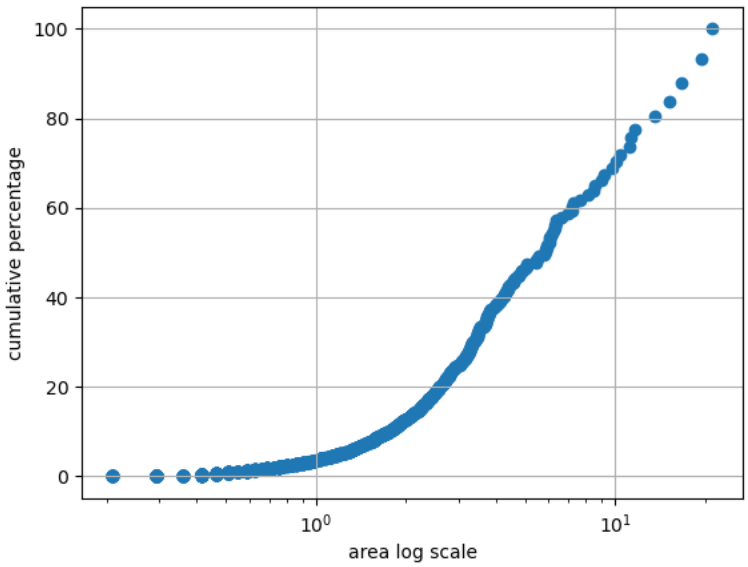

Figure22: Analyzed grading curve for sample shown in Figure - 21

Gradation result of pervious concrete specimens:

Pervious concrete specimens as shown in Figure-23 made of different mixture proportions were analyzed to determine the gradation.

Figure 23: *Pervious concrete specimen (Dimensions: 10cm by 20 cm)*

Grading curve for sample pervious concrete specimen (Dimensions: 10cm by 20 cm) shown in Figure - 24.

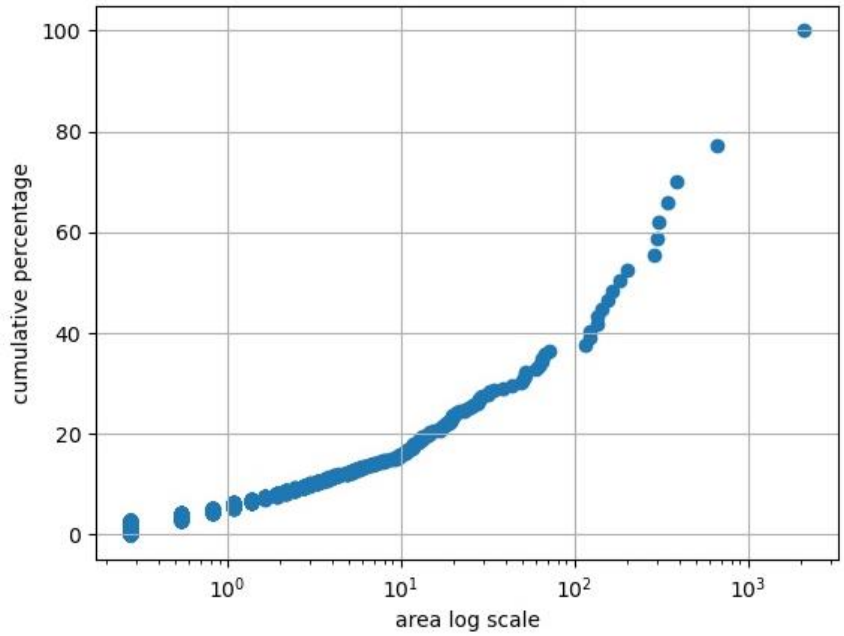

Figure 24: Analyzed grading curve for sample shown in figure 22

4. Conclusion:

In this work, preliminary efforts were made to extract the gradation of pervious concrete specimens and bituminous mixture specimens using the principles of image analysis. The images collected from the laboratory were processed to remove the noise and other unwanted features. The images were analyzed to determine the particle size in 2D plane. The distribution of the particle sizes was similar to that of a typical gradation curve. More improvements are needed to accurately determine the gradation of the specimen without conducting test. It is anticipated that this program will help engineers to determine the gradation quickly and assess the quality of the mixture.

5. References:

- OpenCv.org
- Python.org
- Alasdair McAndrew (2011), Introduction to Digital Image Processing with MATLAB, Cengage learning, India Edition.
- Rafael C. Gonzalez., Richard. E. Woods (2009), Digital Image Processing, Pearson.

Bibliography

- H.M. Zelelew, A.T. Papagiannakis , E. Masad Application of Digital Image Processing Techniques for Asphalt Concrete Mixture Images.
- Ki Hoon Moon, Augusto Cannone Falchetto, Michael P. Wistuba, Jin Hoon Jeong Analyzing Aggregate Size Distribution of Asphalt Mixtures Using Simple 2D Digital Image Processing Techniques
- Aaron R. Coenen, M. Emin Kutay, Nima Roohi Sefidmazgi & Hussain U Bahia Aggregate structure characterization of asphalt mixtures using two-dimensional image analysis

6. Appendix:

Code

```
#images with cropped surface
import cv2 as cv
import sys
from matplotlib import pyplot as plt
```

```
import numpy as np


img = cv.imread(cv.samples.findFile("G:\my_Ml_projects\image processing\p10.PNG"))
if img is None:
    sys.exit("Could not read the image.")


gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
#cv.imshow("Display window", gray)
#cv.waitKey(0)
dimensions = gray.shape
print(dimensions)
print("Height of image",dimensions[0])
print("width of image",dimensions[1])
equ1 = cv.equalizeHist(gray)
clahe = cv.createCLAHE(clipLimit=12.0, tileGridSize=(8,8))
equ = clahe.apply(gray)
#blur = cv.bilateralFilter(equ,5,51,51)
#blur = cv.GaussianBlur(img,(5,5),0)
#ret,th3 = cv.threshold(equ,255,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
ret,th = cv.threshold(equ,170,255,cv.THRESH_BINARY)   #for cement paste
ret,th1 = cv.threshold(equ1,50,255,cv.THRESH_BINARY_INV) # for air voids
ret,th2 = cv.threshold(equ1,0,255,cv.THRESH_BINARY) #whole image
#adding th and th1 will give image for aggregate
dst1 = cv.addWeighted(th,1,th1,1,0)
ret,dst = cv.threshold(dst1,150,255,cv.THRESH_BINARY_INV)
res = np.hstack((th,th1,dst,gray,equ1))
cv.imshow("Display window", res)
cv.waitKey(0)
plt.imshow(res)
```

```
plt.title('my picture')
plt.show()
hist = cv.calcHist([equ1],[0],None,[256],[0,256])
print("Histogram for automatic equalise histogram")
plt.hist(equ1.ravel(),256,[0,256]); plt.show()
hist = cv.calcHist([equ],[0],None,[256],[0,256])
print("Histogram for clahe equalisation")
plt.hist(equ.ravel(),256,[0,256]); plt.show()
hist = cv.calcHist([th],[0],None,[256],[0,256])
print("Histogram for cement paste binary")
plt.hist(th.ravel(),256,[0,256]); plt.show()
hist = cv.calcHist([th1],[0],None,[256],[0,256])
print("Histogram for air voids binary")
plt.hist(th1.ravel(),256,[0,256]); plt.show()
hist = cv.calcHist([dst],[0],None,[256],[0,256])
print("Histogram for aggregate binary")
plt.hist(dst.ravel(),256,[0,256]); plt.show()
n_white_pix_w = np.sum(th2==255)  #whole area
print('Number of white pixels whole:', n_white_pix_w)
n_white_pix_v = np.sum(th1==255)   #voids area
print('Number of white pixels voids:', n_white_pix_v)
n_white_pix_c = np.sum(th==255)     #cement area
print('Number of white pixels cement:', n_white_pix_c)
n_white_pix_a = np.sum(dst==255)  #aggregate area
print('Number of white pixels aggregate:', n_white_pix_a)
#calculating area of each part
speciman_area = 100 * 200
number_of_total_pixels = dimensions[0] * dimensions[1]
mm_square_per_pixel = speciman_area / number_of_total_pixels
area_of_v = n_white_pix_v * mm_square_per_pixel
```

```
area_of_c = n_white_pix_c * mm_square_per_pixel
area_of_a = n_white_pix_a * mm_square_per_pixel
print("area of voids:",area_of_v)
print("area of cement:",area_of_c)
print("area of aggregate:",area_of_a)
area_whole = area_of_v + area_of_c + area_of_a
print ("whole area:",area_whole)
dimensions = gray.shape
print("Height of image",dimensions[0])
print("width of image",dimensions[1])

num_labels, labels = cv.connectedComponents(dst,connectivity=4)

# Map component labels to hue val, 0-179 is the hue range in OpenCV
label_hue = np.uint8(179*labels/np.max(labels),axis=1)
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv.merge([label_hue, blank_ch, blank_ch])

# Converting cvt to BGR
labeled_img = cv.cvtColor(labeled_img, cv.COLOR_HSV2BGR)
sum1 = 0
#for label in labels:
  #  print(label)

# set bg label to black
labeled_img[label_hue==0] = 0
labels = labels.reshape(np.size(gray))
print(np.shape(labels))
label_list = labels.tolist()
label_set = set(label_list)
```

```
total_labels = list(label_set)

for label in total_labels:

   print("area of each label image {} = ".format(label),label_list.count(label) * mm_square_per_pixel)


cv.imshow("Display window",labeled_img)

cv.waitKey(0)

sum1 = 0

# Showing Original Image

plt.imshow(cv.cvtColor(dst, cv.COLOR_BGR2RGB))

plt.axis("off")

plt.title("Orginal Image")

plt.show()


#Showing Image after Component Labeling

plt.imshow(cv.cvtColor(labeled_img, cv.COLOR_BGR2RGB))

plt.axis('off')

plt.title("Image after Component Labeling")

plt.show()
```

end