SUMMER INTERNSHIP

FINDING GRADING CURVE OF VARIOUS PAVEMENT THROUGH IMAGE PROCESSING REPORT

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

SCHOOL OF INFRASTRUCTURE
INDIAN INSTITUTE OF TECHNOLOGY BHUBANESWAR
ARGUL, JATNI -752050, ODISHA
30 JUNE 2020

Introduction:

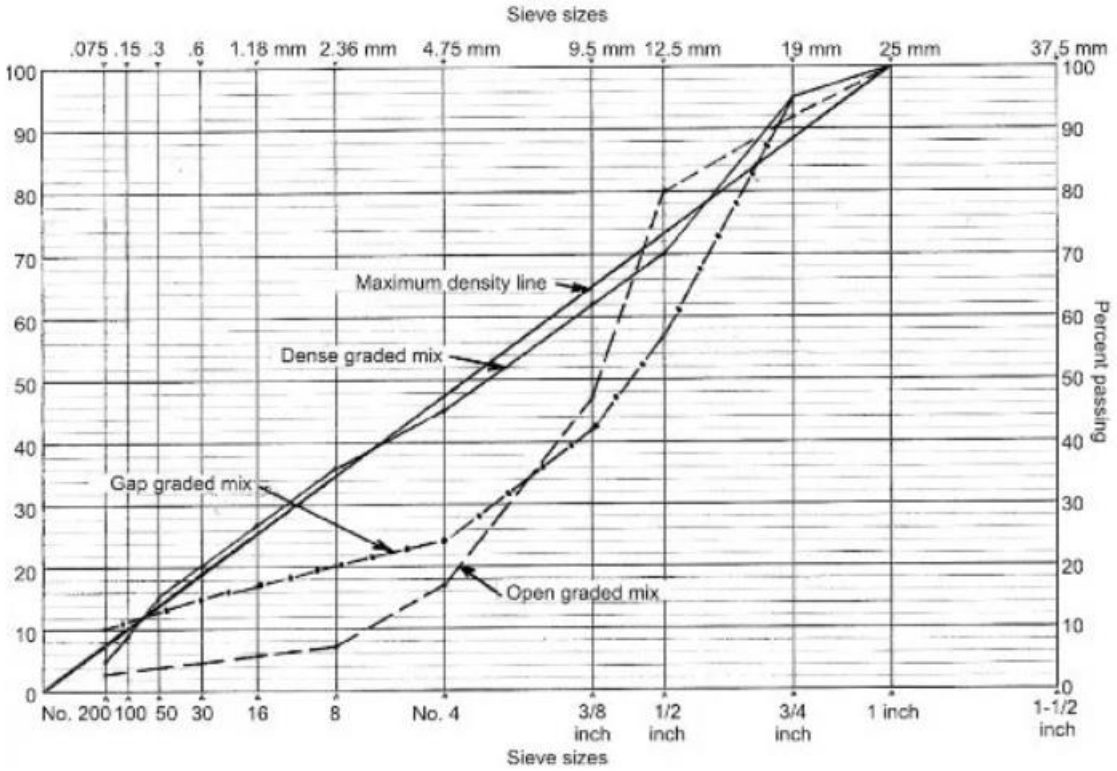
The pavements can be classified based on the structural performance into two, flexible pavements and rigid pavements. In flexible pavements, wheel loads are transferred by grain-to-grain contact of the aggregate through the granular structure. The flexible pavement, having less flexural strength, acts 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 like 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 most desirable characteristics.

Typical 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 provides proper bonding between two layer of binder course and must be thin, uniformly cover the entire surface, and set very fast. 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 into the layer below, plugs the voids, and forms a water tight 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 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 top soil 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:

- Dense Graded Bitumen Mixes

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



The bitumen mix that is densely graded has continuous gradation, say in the proximity of 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 gap graded bitumen mix. The semi-graded mix will have a gradation lying in between the open graded and the gap graded.

Problem Statement:

The aggregate gradation is one of the most important parameters in the mechanical properties of conventional flexible pavement. Thus, determining the aggregate gradation is a very significant subject in civil engineering. Usually, to estimate the aggregate gradation, it is required to separate the aggregate from the bitumen, and this operation can be time-consuming and even dangerous related to chemical solvents. More labour work is involved in separating aggregate from bitumen, so much instrumental loss in cooling the aggregate and doing sieve

analysis of separated aggregate. Moreover, several computer-based methods have been established to model the internal structure of hot mix asphalt (HMA) in two- and three-dimensional methods and can be applied to determine the aggregate gradation, but these methods need special and expensive equipment or so much manual work of entering data. Therefore, 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.

Image processing and analysis:

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, 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.

To study the morphological properties of the mixtures, 2D planar images can be 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

Every image consists of pixels. When image is clicked by the camera it takes value of every pixel in the form of value of RGB. Every code is in between 0-255. Before the analysis of an image is been converted to Grayscale (each pixel has value in between 0-255). Gray scale image helps during removing of unwanted noise and unwanted pixels.

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 is used in entire report.

Methodology:

For cement paste pavement:

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

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 help us.

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

if we see mathematically it add value of R, G, B of every pixel and take an average of it. And give that 0-256 value to the respective pixel.

Grayscale image is not sufficient to get accurate results.

In order to measure the objects in the image, 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. 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.

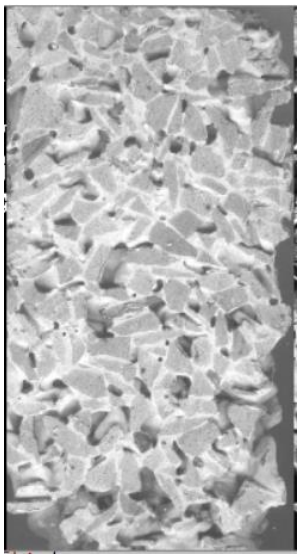


Figure 1: Grayscale image

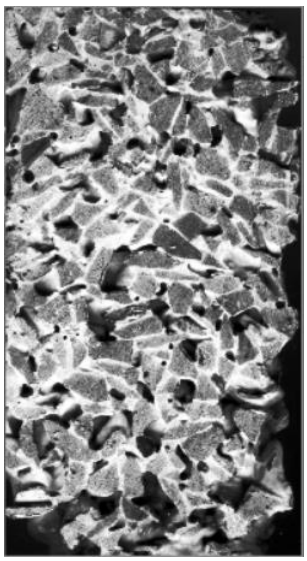


Figure 2: Grayscale image after histogram equalization

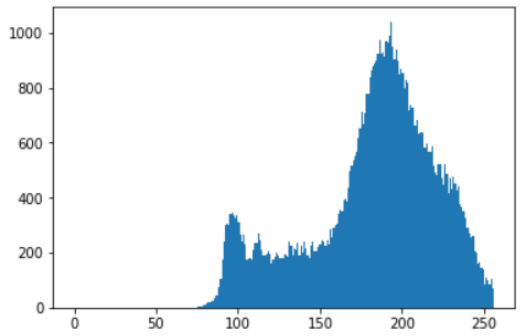


Figure 3: without histogram equalization

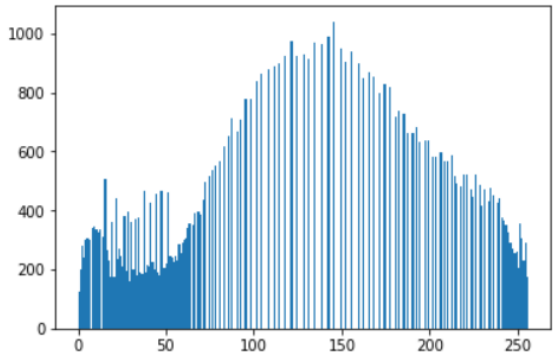


Figure 4: Automatic equalize histogram (Normalization)

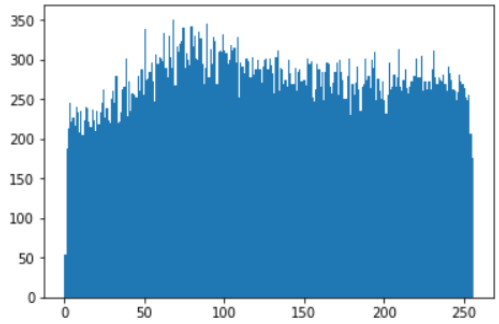


Figure 5: Clahe equalization

- ***Normalization:*** ref Figure 4

equ1 = cv.equalizeHist(gray)

this python code stretches the histogram to equalize the pixel values over the range. **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

$I_n = (I - Min)\frac{newMax - newMin}{Max - Min} + (newMin)$

I_n = New intensity value of pixel

I = original pixel intensity value

Adaptive histogram equalization: ref 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. So, 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, 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. But 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 doesn't consider whether a pixel is an edge pixel or not. So, it blurs the edges also, which we don't 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. So, 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.

```
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.

cv.threshold(equ1,50,255,cv.THRESH_BINARY_INV) due to 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 6: Cement paste

Figure 7: Air Voids

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 8: Aggregate image

After getting the pre processed image. We can proceed towards analysis for grading curve of the aggregate.

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:

Previously, we got three different images for cement paste, Aggregate and Air voids. In each image white pixels shows cement paste, Aggregate and Air voids.

Following python code help in finding respective areas.

```
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
```

When image got converted into binary then we only have pixels with intensity values 0(black) and 255 (white). Measuring total number of pixels having intensity value 255 gave number of white pixels. Using geometric scaling total area of those white pixels has been calculated.

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

```
dimensions = gray.shape
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 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:

- **Connected component:**
  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.

First pass:

For each non-zero pixel, we check its neighbours.

If it has no non-zero neighbours — we know it is a new component — so we give it a new label.

If it has one non-zero neighbour — these pixels are connected — we give it the same label as the neighbour.

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

The neighbours all have the same label. So, we give the current pixel the same label.

The neighbours have different labels. This is the tricky part. We know all of these pixels are connected now, so the labels should be all the same. The classical approach to solve this is the following. We set the current pixel to the neighbours' lowest label. Then we also keep a note of the equivalences of all the connected labels — that is, which different labels should actually be the same. We will solve these on the second pass!

Each non-zero pixel will now have a label. However, some connected regions will have different labels. So we need to go over the image one more time to solve this. We do this by using our recordings of 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.

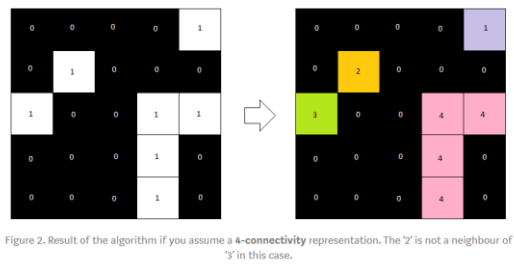


Figure 2. Result of the algorithm if you assume a **4-connectivity** representation. The '2' is not a neighbour of '3' in this case.

Figure 9: 4 connectivity representation

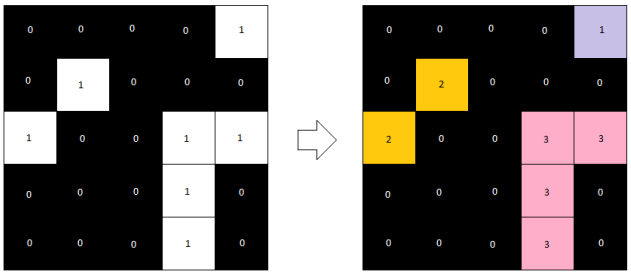


Figure 3. Result of the algorithm if you assume a **8-connectivity** representation. The "2's" are neighbours.

Figure 10: 8 connectivity representation

Pseudo code for Connected component labelling:

```
algorithm TwoPass(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.

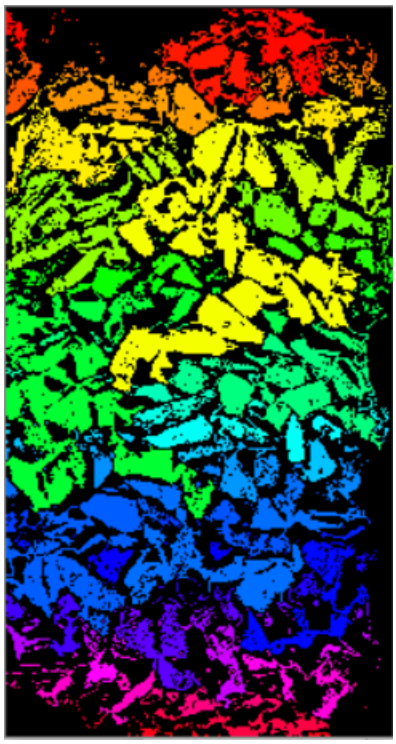


Figure 11: Aggregate classification using connected component method

For bitumen pavement:

Bitumen Pavement has only two components.

1. Bitumen
2. Aggregate

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

This will convert RGB image into grayscale.

Figure 12: Original image

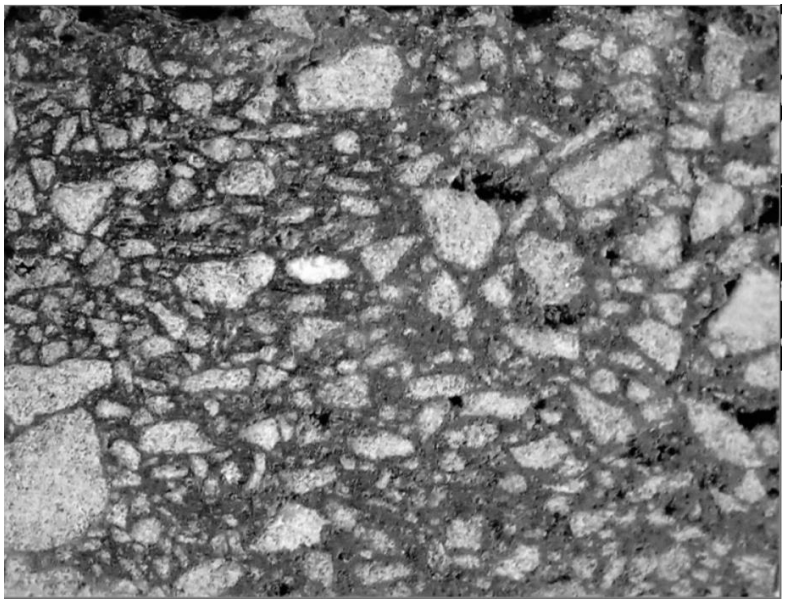


Figure 13: Grayscale Image

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)
```

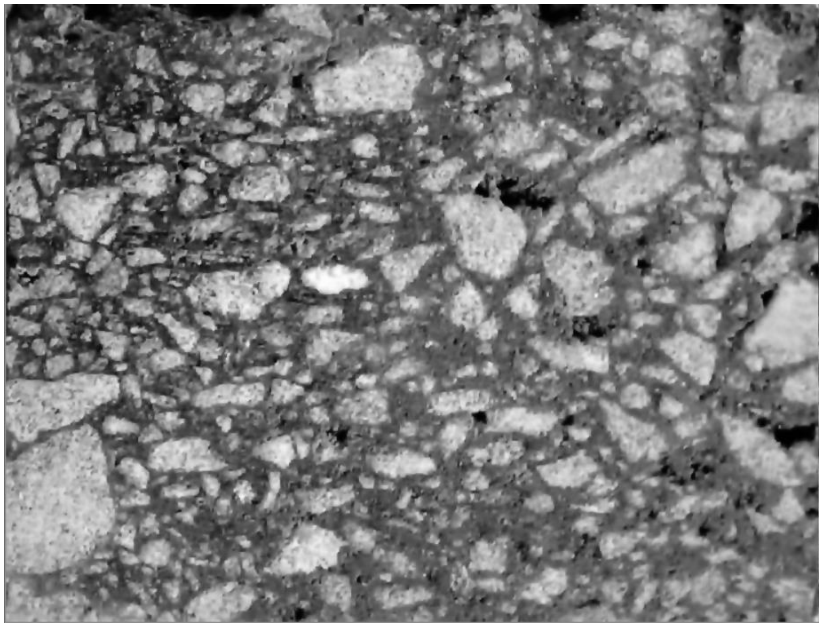
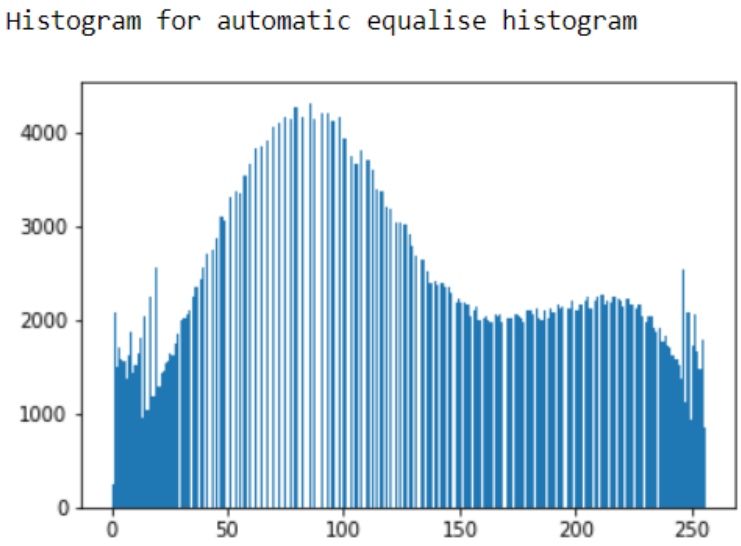


Figure 14: Image after applying bilateral filters

Same like cement paste sample, histogram stretching equalization is applied after filtering the image.



Before going for the analysis image has been converted to binary using following code.

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

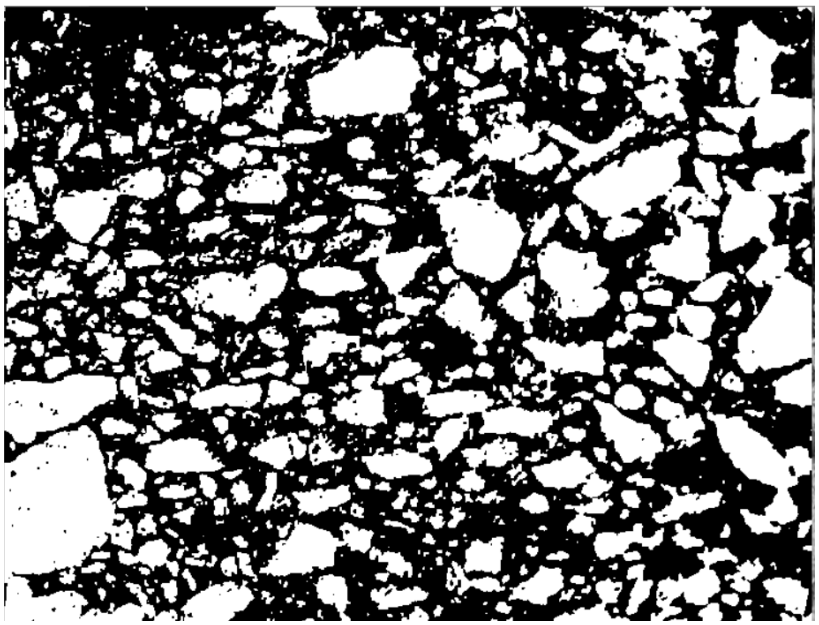


Figure 15: binary image

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.

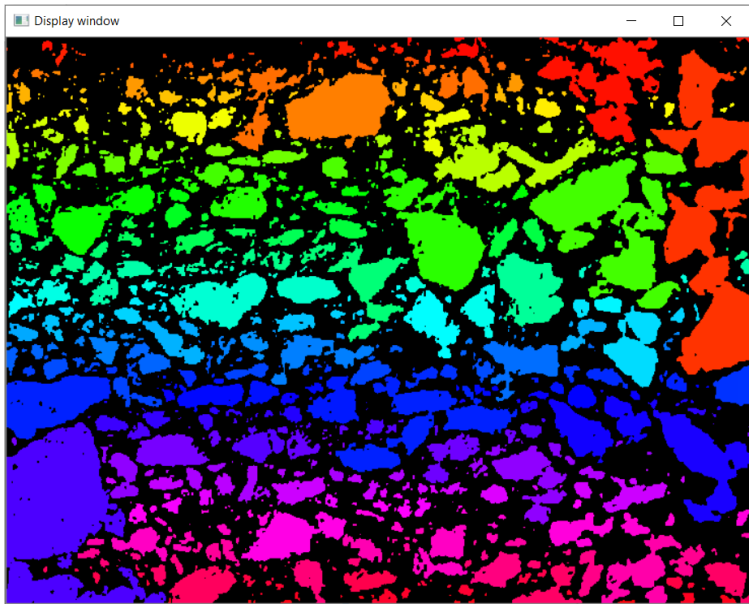


Figure 16: Image after connected component analysis

Result and analysis:

Gradation result of bitumen sample –

Preliminary results:

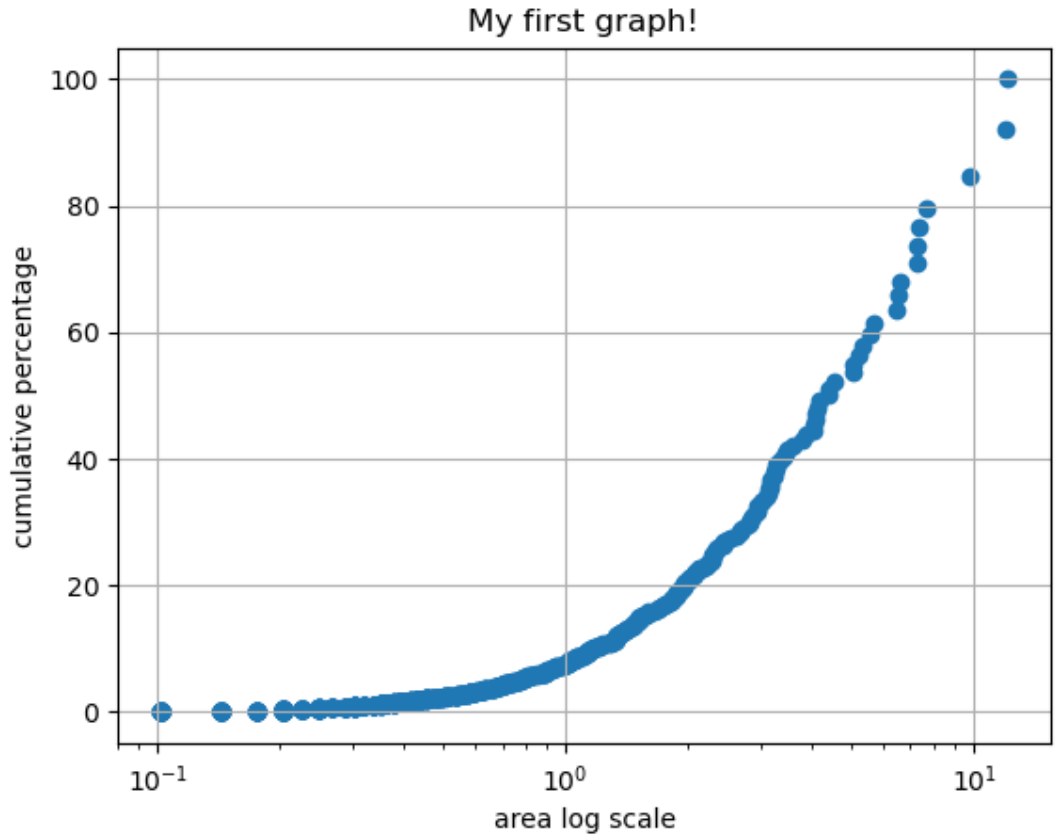


Figure 17: analysed grading curve for bitumen sample 7 by 6

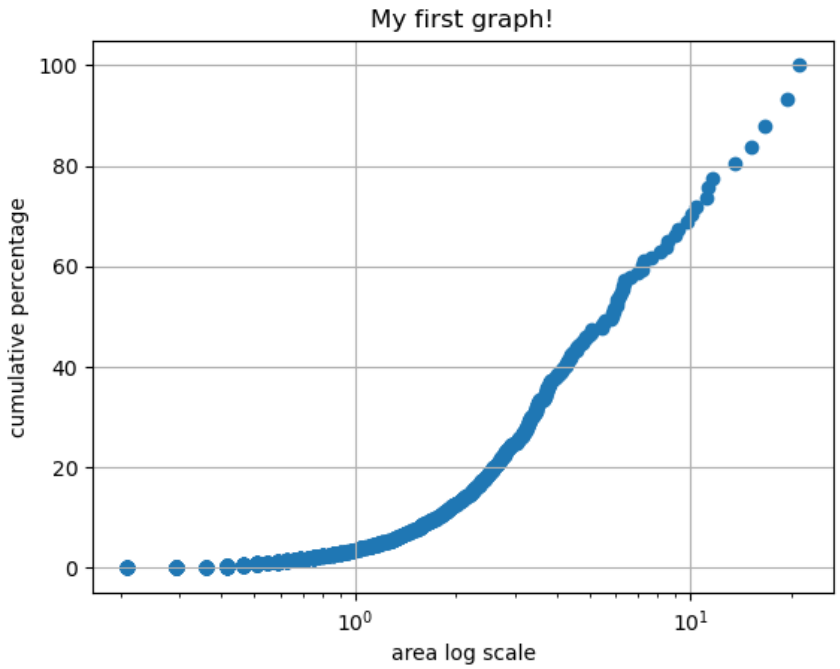


Figure 18: Analysed grading curve for sample 30 by 5

Gradation result of bitumen sample –

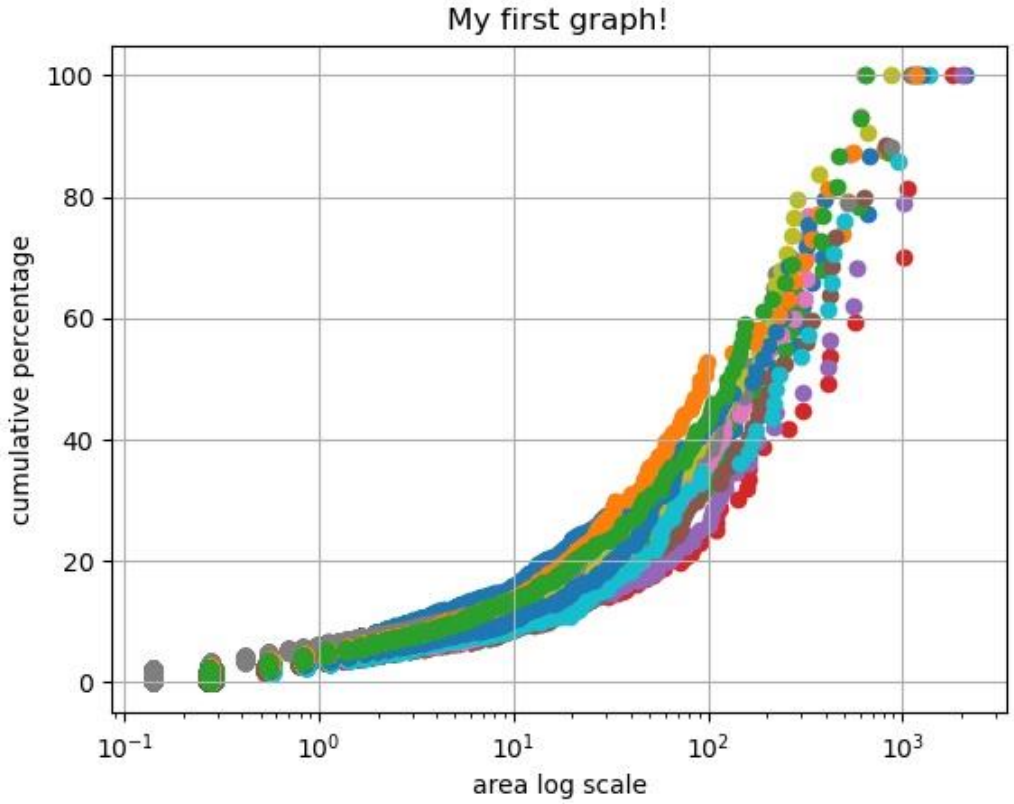


Figure 19: Analysed grading curve for 11 sample of cement paste pavement.

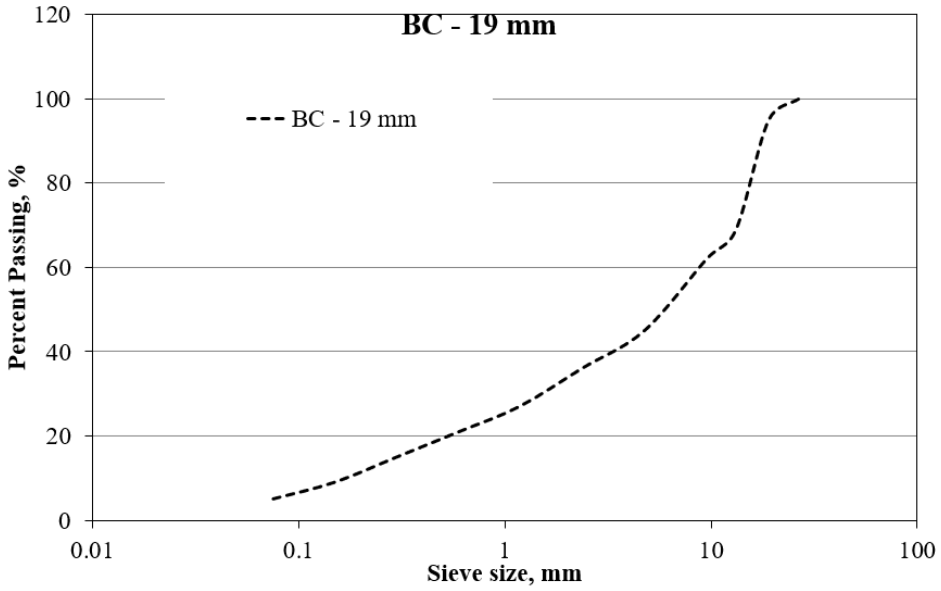


Figure 20: Practical grading curve for both the bitumen sample

Conclusion:

We got a result mostly same with the practical grading curve. Practical results and data extracted through image processing are mostly same. both has shown the grading is **Gap graded Mix**.

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.
- H.M. Zelelew, A.T. Papagiannakis , E. Masad Application of Digital Image Processing Techniques for Asphalt Concrete Mixture Images.

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
```

Submitted by:

Pranav Baradkar

3rd year undergraduate at IIT Bhubaneswar

Guide:

Anush K.Chandrappa, Ph.D.[IIT KGP]

Assistant Professor

School of Infrastructure

Indian Institute of Technology Bhubaneswar