

# Portable Handheld by OCR Signal Generation

Shaigi Ramesh PT

## ABSTRACT

Vision is considered to be one of the most essential of human senses and plays pivotal role in our life. Appallingly 285 million people in the world are visually impaired and about 9 out of 10 blind children in developing countries like India have no access to education. The current techniques and technologies fail to fully address the need literacy need of these disadvantaged ones. Each of these technologies, be it a simple printed Braille books or a more sophisticated PC based e-reader have their own shortcomings, like availability, portability, high cost, need for expertise and so on.

Theia, also known as the Greek goddess of vision and sight, is an embedded system device that provides ubiquitous computing for people with no or low vision. With the tagline "Reading within Reach," Theia is small, portable and handheld. It is a multimodal e-reader that reproduces text (handwritten or printed) into 1) refreshable array of 8-celled Braille and/or 2) synthesized audio {via either a) the top or back of the device or b) Theia as an iPhone case}. The CCD camera chip of the device captures the text, and OCR & signal generation software bundle processes the required data.

## IMAGE MANIPULATION

### ABSTRACT AND MOTIVATION

Image manipulation plays a vital role in the entertainment industry and a well pronounced role in the scientific domain such as remote sensing and astronomy. often, while filming movies the shot requires intricate details that can be produced real time and also certain shot with no real world connection to based on "Blue screen technology" helps in this aspects to capture a completely unrelated motion, such as of an actor and suppressed it quite completely on a complexity new image. it is done by capturing and determining the R,G,B manual of the back drop and running a system to effectively mask only at value of background. Thus effectively removes the foreground, which can be used around in a myriad of ways.

The extracted foreground can be "mirrored" by just filling the values along fixed column boundary. the less use of a such a extracted image is inserted in such complex background.

One of the other image manipulation technique is "Colour manipulation" where is an color image converted into a myriad of forms for the cause of aesthetia values. it can be converted into white image, grayscale image or more famous "sefia image, sefia image gives a rustic feel and is prepared by photo enthusiast.

Here we implement each of these technique and concatenate the final image for the sake of appreciation of techniques. with given a value scanned harry potter image and destinate, larger castle image, we go about applying the image manipulation techniques.

### APPROACH AND PROCEDURES

The first part of the problem deals with retrieval of foreground image of the harry potter from the green background image.

For the we need to determine the RGB values of the background image. but due to different lighting, shadow and other perturbations the background value does remains constant but varies by a range of RGB value.

This procedure is as follows.

Scan the words of image and find the average value of RGB, which consists of

$$R=168$$

$$G=231$$

$$B=30$$

Next step is to find deviation around the values of RGB, The majority of deviation curve is (168,231,30) and the values contains the background to select the range find a cut off percentage, which here is 85 percentage and then range turns outlier.

$$R(163,191)$$

$$G(210,235)$$

$$B(27,108)$$

For all the values between these range the finites are turned to  $R=G, B=255$  and then a white background

Next, the mirror the harry potter image horizontally can be done in really simple way, which is

- New image [row][column][colour values] = original image [row][boundary columns][colour values]

This flips the image horizontally.

Next, the image insertion runs of the conditions algorithm that make sure only the foreground of the harrypotler image sticks on the [600,100] position of the castle.

- Harrypotler[row][column][column value]!=255
- CastleData[row+600][column+100][colour value]=harrypotler[row][column][colour values]

Else no insertion.

So, by running this iteration for the entire final of harrypotler data we can achieve the insertion.

Thus we do sephia filtering by applying the formula, of

$$\begin{aligned} [SF(k)] &= [0.393 \ 0.762 \ 0.189] [I(R)] \\ [SF(g)] &=> [0.349 \ 0.686 \ 0.168] [I(G)] \\ [SF(b)] &= [0.272 \ 0.534 \ 0.131] [I(B)] \end{aligned}$$

And for  $SF > 255 \Rightarrow SF = 255$ , we make sure the gray values as the colour value point doesn't exceed the maximum and flip value. Once we obtained all the said images concatenation is carried out between the mirror inserted castle image and the sephia filtered image which again exceedingly easier, with conditional insertion into a bigger image being executed.

So column, column image boundary

Concatenated image [row][column][colour value] = mirror image [row][column][colour value]

Else,

Concatenated image [row][column][colour value] = sephia [image][row][column] - boundary value [colour value].

Iteration for all pixels of concatenate image. This way interleaved raw images with image manipulation are created.

## EXPERIMENTAL RESULTS

Shown below are the results for Problem 1:



Figure 1: Original image with green background

Figure 2: Image with green background removed



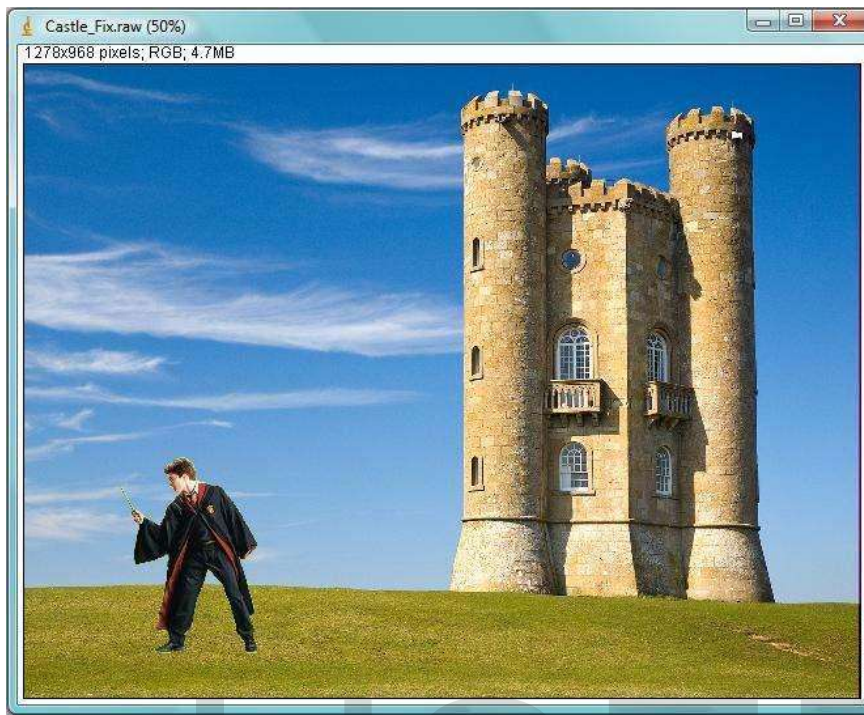


Figure 4: Insertion of Harry Potter image in the Castle image at point [600,100]



Figure 5: Mirrored image of the Castle-Harry Potter combination image





Figure 6: Sepia transformed Castle-Harry Potter combination image



Figure 7: Sepia transformed and Mirrored Castle-Harry Potter combination concatenated final

## DISCUSSIONS

Harrypotler background cleaned image in figure 2 shows the adaptively removing background is the most effective and efficient method comparing it to the original image shows the foreground has been completely sequestered out. for better performance the cut of percentage can be reduce or rather than increased from 85 percentage to more

But this should create the problem of removing essential part of foreground.

So, a trade off exists between effective background removal and conserving the foreground details.

The mirrored image and castle image insertion can further achieve algorithm efficiency by combining the iteration together in a single champ

The concatenated image shown how different sepia and original image in aspect and allows excellent demonstration of image manipulation technique.

## HISTOGRAM EQUALIZATION

### PART A: IMAGE ENHANCEMENT(Gray scale image linear scaling and Histogram equalization)

## ABSTRACT AND MOTIVATION

Contrast enhancement is an aesthetic quality improvement of an image and deals with making a image more appearing usually. when the grayscale value of the image or rather concentrated in gray scale value of image or rather concentrated in grayscale segment, the image tends to appear unusually. apart from H, the image elements become hard to distinguish and go their need for image enhancement.

The contrast enhancement can be done in two positive way "linear scaling" and "histogram equalization". linear scaling deals with sketching and concentrated grayscale errors as entire or rule group of entire range. this preserves

the grayscale distribution but produces irreforable contains effect, limits nevertheless easier and faster. Histogram equalization deals with distributing the grayscale value equally to all the range. it is bit effective but gives a really good output image with high aesthetic value.

## APPROACH AND PROCEDURES

The problem deals with three grayscale images with each having their grayscale value centered around low, middle and high greyscale value respectively.

Each of these images are to be processed with contrast enhancement technique of with linear scaling and histogram equalization.

Firstly taking linear scaling, we need to find out and decide a range of grayscale value which be extended to complete grayscale range.

This can be done by two methods

1. placing a percentage cut off value around the peak to determine the range, which is most effective but computationally intensive.

2. Have fixed range and let it slide such that the midrange of its peak value.

Looking for us, in the given problem the grayscale value are existent and concentrated only in a certain small range. so we, forever third option of finding lowest non zero grayscale value,  $F_{min}$  and height nonzero grayscale value  $F_{max}$ .

For every grayscale histogram is found out such that complete image is scanned and frequency of occurrence of each fixed value is recorded in a 1-D array  $Histmat[255]$ . this is iterated for complete grayscale value and first non zero occurrence.  $F_{min}$  and last non zero occurrence,  $F_{max}$  is found.

➤  $Histmat[i], i[0,255]$

If (Histmat[previous]==0 and  
Histmat[present]!=0=>Fmin=i

If (Histmat[present]!=0 and  
Histmat[previous]==0=>Fmax=i

With fmax and fmin found full range linear filter can be  
applied as

$G = G_{min} + G_{max} - G_{min} / f_{max} - f_{min} + \text{image data } I[\text{fixed value}]$

Here  $G_{min}=0, G_{max}=255$

Thus in this way linear scaling for each image is done and  
this corresponding transfer function and histogram plotted.

Histogram equalization involves spreading out the entire  
grayscale value in a equality manner to conflict  
range, creating a least perceptile image needed. it involves  
first finding out the gray scale breakfull which depends on  
the image size and determines the occupability of  
individual gray levels.

For image of size,  $M*N$

The grayscale breakup is given by  $M*N/255$

Now the entire image is scanned and iterated for whole  
grayscale value [0,255] with first one lowest gray scale  
occurrences mapped to zero till gray scale break up is  
satisfied and then as to next

This simplified with help of simple incremental variable

- Gray scale [0,255]
- Image pixels

Next data[pixels value] = current grayscale

If (count < scale break up)

Count increment;

Else reset count and increment current gray scale,

Thus the three given images are historically equalized and  
enhanced.

## EXPERIMENTAL RESULTS

Shown below are the results for Problem 2, Part A:



Figure 1: Original grayscale images with grayscale values of pixels concentrated in low, middle & high range respectively



Figure 2: Full range linear scaled images of rose dark, mid & bright respectively



Figure 3: Histogram equalized images of rose dark, mid & bright respectively

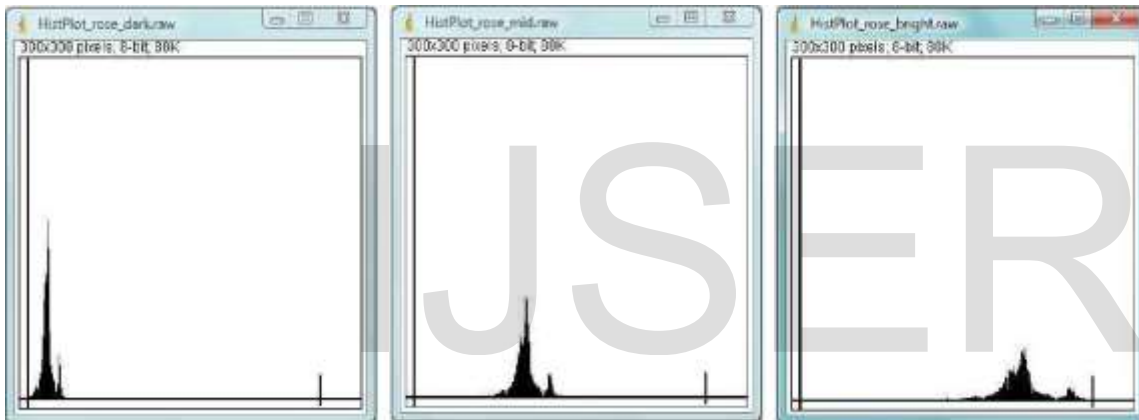


Figure 4: Histogram plot of rose dark, mid & bright respectively

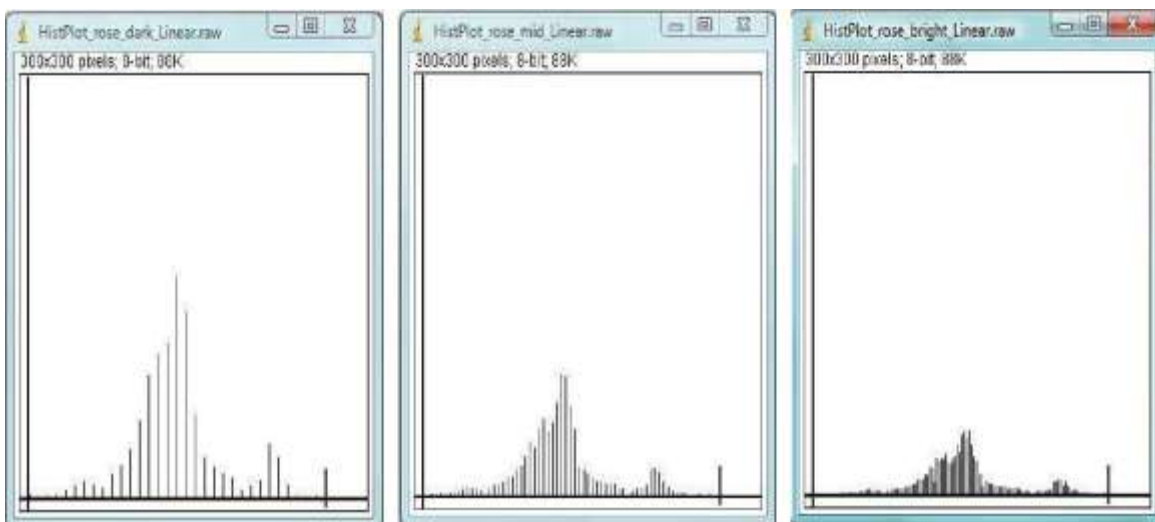


Figure 5: Histogram plot of linear scaled rose dark, mid & bright respectively

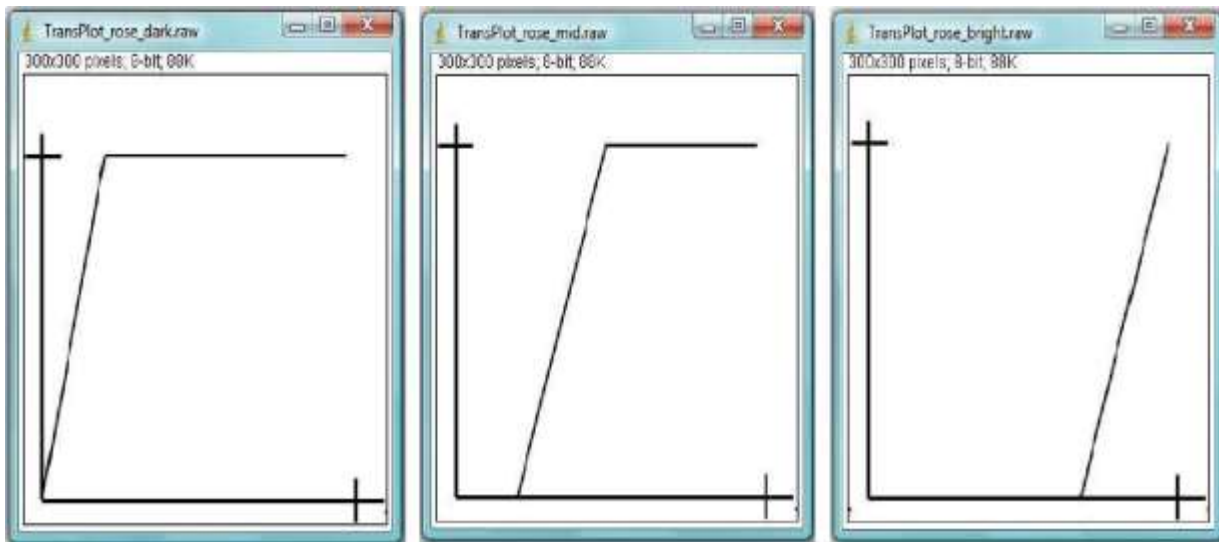


Figure 6: Transform function plot of linear scaling of rose dark, mid & bright respectively

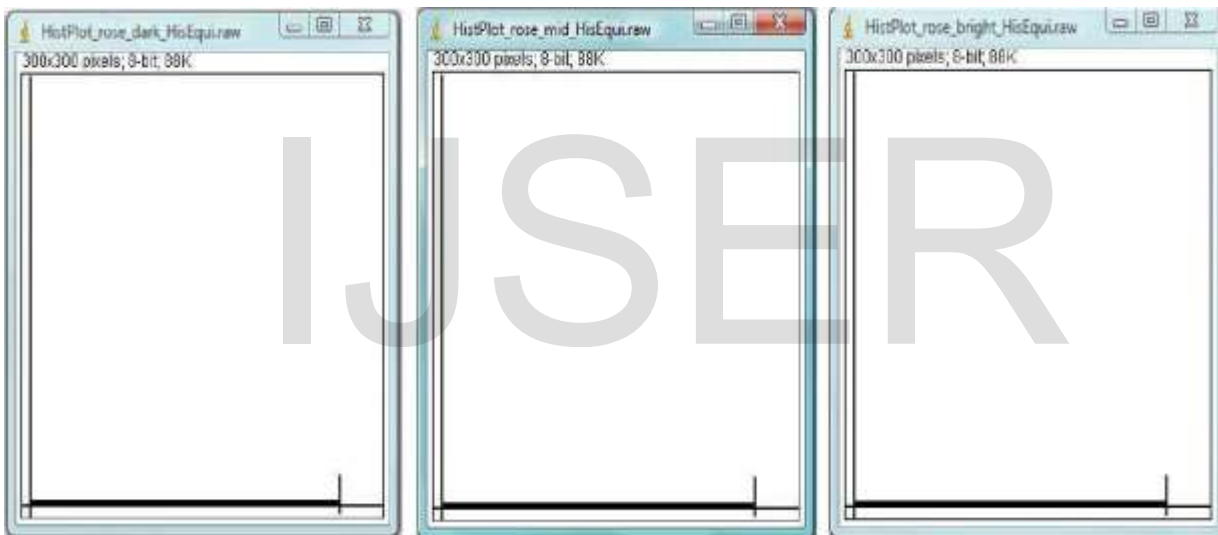


Figure 7: Histogram plot of histogram equalized rose dark, mid & bright respectively

## DISCUSSIONS

From figure 2 (linear scaling ) and figure3 (Histogram equalization).it is completely obvious that the contrast enhancement does in prove usual and aesthetic activity of the image. But a comparison between figure 2 and figure 3 themselves paint a different picture ,that clearly shows histogram equalization producing a more aesthetic and visual image and full range linear scaled one and also

there's perceptible valid in certain elements in the linear scaled image

From the histogram plot of linear scaled image .it is quite obvious that it suffer from the continuous effect ,which splits out the inherent state the original image histogram across varied largely gray scale values.this reduces the smoothness of the image and makes it more jithery and visually a bit less unaffectly.that is why histogram equalization is preferred alternative



## PART B :COLOUR IMAGE HISTOGRAM EQUALIZATION (Histogram equalization in RGB and HSI colour scale)

### ABSTRACT AND MOTIVATION

Histogram equalization is considered as the last method for image enhancement. histogram equalization for a gray scale image is really a simple and effective method producing great results. a trait applying the same technique to colour images as such treats as certain and undesirable quality corruption, especially when applied adhoc to images in RGB colour space. an alternative approach is to transform RGB colour space to HSI colour space and then carry on with equalization of intensity value. this is the effective technique as it is the intensity component of the image that constitutes to visually activity, with the and saturation conserved and reproduce in RGB colour space, the image preserves it originally and its contrast improved at the same instance.

### APPROACH AND PROCEDURES

This first part of the problem is to apply histogram equalization to each of the colour space of RGB separately and independently and combine them together for the final image in the interleaved format

For all iteration of R,G,B in original source

- Grayscale[0 255]
- Image pixel

If (image data[pixel[colour value]]==grayscale)

New data[pixel][colour value]= current grayscale

Conditional applicability of scale break up and gray scale decrement.

The scale breakup, as discussed earlier given a measure of the pixels to be equated to the gray level and is depended on the image dimension.

If image dimension is  $M*N$

Scale break up =  $M*N/255$

The final image is then reproduced in the interleaved RGB value format and then achieving histogram equalization in the RGB colour space.

Next, we deal with histogram equalization in HSI colour space and reproduction the image. HSI is an alternate way to represent an image and manipulate it an easier way RGB to HSI is converted by the following way.

Histogram equalization is applied to the intensity value.

Step 1: Normalization

- $R/R+G+B = r$
- $G/R+G+B = g$
- $B/R+G+B = b$

Here we have to make sure R,G,B not equal to zero, if so then breakout and equate each of HSI to zero. also r,g,b should be float variables to allow the room for decimal values which hold some more information.

Step 2: HSI calculation

1. S calculation

$S = 1 - 3 \min(r, g, b)$

The minimum function as to be predefined and its value can be conserved as such with no conversion.

2. I calculation

$I = (R+G+B)/3$

For the histogram equalization it is necessary to how  $I[0,255]$ , to go actual I can be calculated.

As,

$$I=(R+G+B)/3*100,I[0,255]$$

3.H calculation

If  $b \leq g$

$$H=\cos^{-1}(0.5[r-g+r-b]/[(r-g)^2+(r-b)(g-b)]^{0.5})$$

Where  $H [0,3.14]$

Else,

$$H=6.28-\cos^{-1}(0.5[r-g+r-b]/[(r-g)^2+(r-b)(g-b)]^{0.5})$$

Where  $H[3.14,6.28]$

The whole value is iterated for entire image pixel and stored respectively in a 1-D flat array each.

Histogram equalization is then applied to I value and anew data array is generated.

To reproduce the image again in RGB colour space,transformation is done on HSI Values.

Step 1:Parametric value calculation

Convert I into decimal normalised value

$$I[\text{pixel value}]= I[\text{pixel value}]/255$$

$$X=i(1-s)$$

$$Y=i(1+s\cos(h))/\cos(3.14/3-h)$$

$$Z=3i-(x+y)$$

Step2:RGB substitution

$$\text{If}(h < 6.28/3)$$

Then

$$b=x, r=y, g=z$$

$$\text{if}(6.28/3 < h < 4*3.14/3)$$

then

$$r=x, g=y, b=z$$

$$\text{if}(4*3.14/3 < h < 6.28)$$

$$g=x, b=y, r=z$$

step3:Gray scale expansion

$$R=r.255$$

$$G=g.255$$

$$B=b.255$$

Thus RGB image is produced

## EXPERIMENTAL RESULTS

Shown below are the results of Problem 2, Part B:



Figure 1: Original church window image



Figure 2: Church window image with RGB colour space and HSI colour space equalization

## DISCUSSIONS

There is a marked difference between the direct RGB equalized and I equalized image. The direct RGB equalized image has more of colour disparity and filters due to the inflexibility of actual colour and equalized colour at pixel of large inflection where one other hand I equalized image is smooth colour wise and list of eye but it also suffer at point from colour inflection on the comparative note, I equalized image is more suitable than RGB equalized as it is the intensity of the image and majority decides contrast variation and consequently the image aesthetics.

## EDGE DETECTION

### Part(a): Edge Detection, Basic Edge Detection

#### I. Abstract and Motivation

It becomes an imperative in certain situation that only the outer pattern of the given image is needed. Such would be the cases as in pattern recognition, image segmentation and so on. This can be done by the process of Edge Detection, whereby the outer edges of the image are delineated and presented in a binary form for further processing and manipulation. The edge detection technique is based on detecting in the gradient changes along the image and then separating the image components out on a thresholding basis.

There are many algorithms to perform the process of edge detection, the major ones used in this section are 1) first order edge detection, which measures and uses single pass image gradient and 2) second order edge detection, which measures and uses double pass or gradient of image gradient. The quality of the edge obtained depends on many parameters and is heavily affected by them. The parameters are the level of thresholding, the mode of calculation of image gradient, the image content as such and the mainly the image detection technique. Our motivation is to study the effects of these parameters and fine-tune them to get a better result. Also, find out the best algorithm for the process of edge detection. The presence of noise in the input image heavily affects the edge detection process and we are to study such effects and if we can remedy it.

#### II. Approach and Procedures

The process of edge detection is manifold which basically revolves around the axiom that the point on the image where the edge is supposed to exist has high level of gradient difference from the rest of the points in the image. So, the different types of edge detection revolve mainly around concept of developing better gradient detection and how effectively to use the calculated gradient. Depending on the way that the gradient generation is implemented and the image gradient is used, the edge detection is broadly classified into 1) First-order Derivative Method and 2) Second-order Derivative Method with Zero Crossing

#### (i) First-order Derivative Method

The basic step in all the first order derivative method

- 1) Find the row and column gradient of the image by applying any of the given masks

$$G_R(j,k) = \frac{1}{K+2} [(A_2 + KA_3 + A_4) - (A_0 + KA_7 + A_6)]$$

$$G_C(j,k) = \frac{1}{K+2} [(A_0 + KA_1 + A_2) - (A_6 + KA_5 + A_4)]$$

- 2) the combined image gradient and store it for processing



```
ColumnGradient[]=ImageData[matrix_index]*ColumnGradient[matrix_index]
```

3) Fix a threshold for segregating edge points and non edge points, and scan the complete gradient image. Compare the gradient with the set threshold and determine the edge and non-edge points.

```
E(j,k)= 1; G(j,k)>Threshold
E(j,k)= 0; G(j,k)<Threshold
```

The difference between the number of First order edge detectors is the mask that is used to calculate the image gradient. The output of the image varies heavily depending on the threshold selected, so it forms a key differentiating parameter here.

So, we devise here a method to “adaptively determine the grayscale threshold value” for the given image from the percentage of the image part we consider to be non-edges. This improves the edge detection process dramatically for the given input image.

### Roberts

The mask for the Roberts is given as

Row gradient	Column gradient
$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

The process is:

1) Calculation of the row gradient by applying the row gradient mask to the pixel in consideration.

For all the image points

```
RowGradient[]=ImageData[matrix_index]*RowGradient[matrix_index]
```

2) Calculation of the column gradient by applying the column gradient mask to the pixel in consideration.

For all the image points

3) Calculation of the combined image gradient from the row gradient and column gradient.

For all the image points

```
ImageGradient[]=sqrt(ColumnGradient[]^2 + RowGradient[]^2)
```

4) Using the given gradient map and the percentage of thresholding deemed necessary, we calculate the grayscale threshold value for the image adaptively

The count of grayscale values below given level:

```

For all the gradient values starting from 0
  For all the image points
    If
      ImageGradient[] < CurrentGradientValue
        Count++

```

The conversion count value to percentage

PercentageCount = Count \* 100 / 256 \* 256

Percentage comparison and threshold determination

```

If PercentageCount > GivenPercentage
  Threshold = CurrentGradientValue

```

**Frei-Chen:**

The quality of the output image varies widely depending on the percentage of thresholding and the implementation of values from 80 to 100 gave a myriad of outputs. The value of 92% seems to give the best output value for the edge detection.

#### (i) Second-order Derivative Method with Zero Crossing

It can be applied in two ways, either as 1) LOG, Laplacian of Gaussian or 2) DOG, Difference of Gaussian, which approximated to LOG when correct parameters are chosen and it helps in faster and more efficient implementation of process. The edge detection process here is severely afflicted with noise, so we apply Gaussian smoothening first to remove out the noise from the input image, then the cascaded Laplacian function is applied and the edge detection process carries on from there.

The base parameters to be taken care in the in the application of the second order derivative are:

5) The entire image is swiped with the found out threshold value and generating edge image in the process

For all image points

If ImageGradient[] < Threshold

EdgeData[] = 1

Else

EdgeData[] = 0

The same technique is applied for the other filters and the result is obtained.

**Prewitt:**

Row gradient	Column gradient
$\frac{1}{3} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{3} \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

**Sobel:**

Row gradient	Column gradient
$\frac{1}{4} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{4} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

The window size, W

The Standard deviations, S1, S2

The Cut-off Parameter, C

Which are related to each other as,

$$W \geq C$$

$$S1 = 2\sqrt{2} C$$

$$S2/S1 = 1.6$$

The DOG operator is given as,

$$H(x, y) = g(x, s_1)g(y, s_1) - g(x, s_2)g(y, s_2)$$

The complete process is given as,

Creation of Gaussian matrix for the given

window size and standard deviation. The

Gaussian function is given as:

1) Creation of the DOG operator from the given window size, standard deviations and using the Gaussian matrix

For all the window elements

$$H[] = (g(a,s1)*g(b,s1)) - (g(a,s2)*g(b,s2))$$

3) Boundary fill the given image and create the gradient map by applying the DOG operator matrix on it.

For all the image elements

GradientImage[] = (ImageData[Windowindex]\*H[Windowindex])

4) Adaptively find the lower and upper threshold for the gradient image from the percentage in consideration.

The count of grayscale values below given level:

For all the gradient values starting from 0

For all the image points

If

ImageGradient[] < CurrentGradientValue

Count++

The conversion count value to percentage

$$\text{PercentageCount} = \text{Count} * 100 / 256 * 256$$

Percentage comparison and threshold determination

If PercentageCount > GivenPercentage

Threshold = CurrentGradientValue

In such a way two threshold is found LowerThreshold and UpperThreshold.

5) Apply the thresholds to the Gradient image in such a way that the values below the LowerThreshold becomes -1, above the UpperThreshold map becomes +1 and in between it 0. This creates the Zero Crossing map.

If GradMap[] < LowerThreshold

CrossMap[] = -1

if

GradMap[] < UpperThreshold

UpperThreshold and If

GradMap[] > LowerThreshold

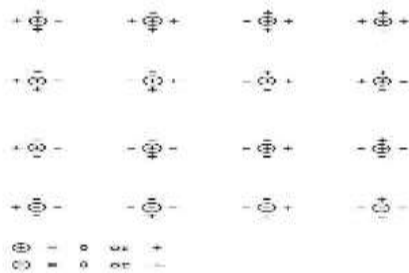
UpperThreshold

CrossMap[] = 0

If GradMap[] > UpperThreshold2

CrossMap[] = 1

6) Find the zero crossing points for the Zero Crossing Map and label it accordingly as edge in the output binary image. The zero crossing is done by the use of patterns as given here:



The value of change in Lower and Upper threshold makes only a slight change in the quality of the output image and this was checked by varying values and optimum one was found to be at 10% and 90%

### III. Experimental Results

Following are the results for the problem 3(a):



Figure 1: Original house image

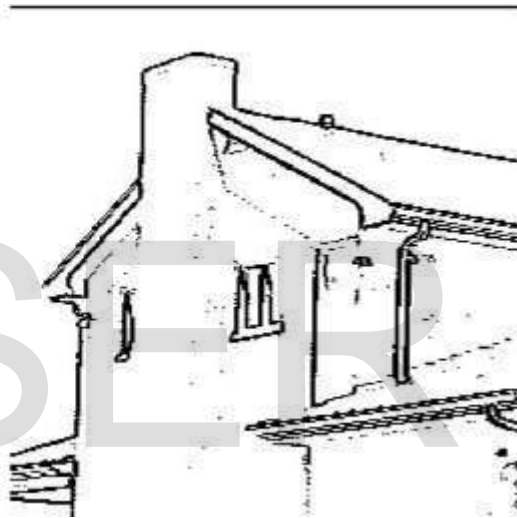


Figure 2: Roberts Filtering on Original house image

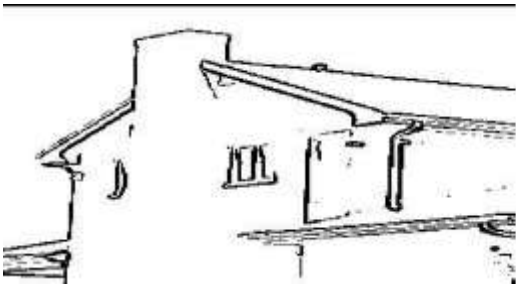


Figure 3: Prewitt Filtering on Original house image

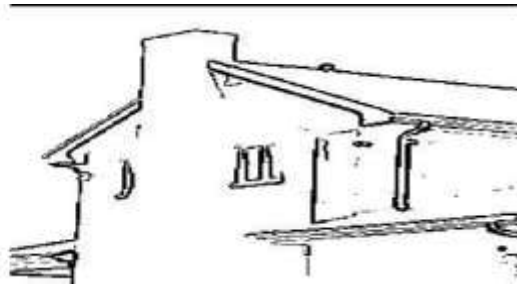


Figure 4: Sobel Filtering on Original house image



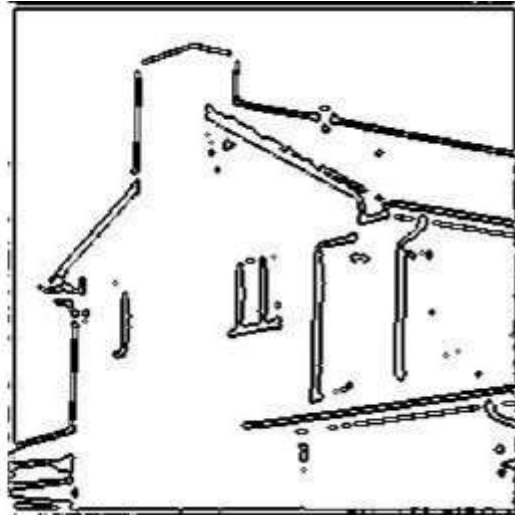
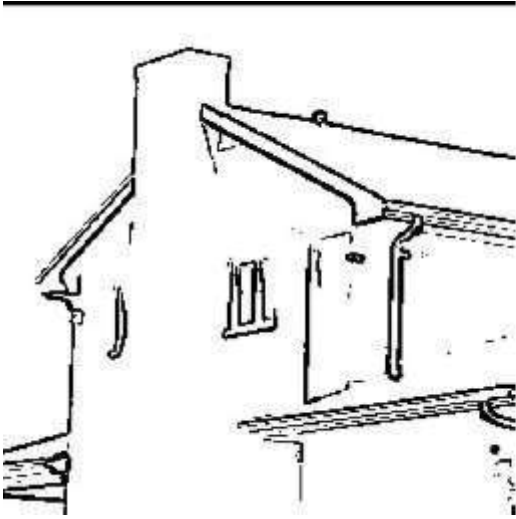


Figure 5: FreiChan Filtering on Original house image Figure 6: DOG window 11 Filtering on Original house image

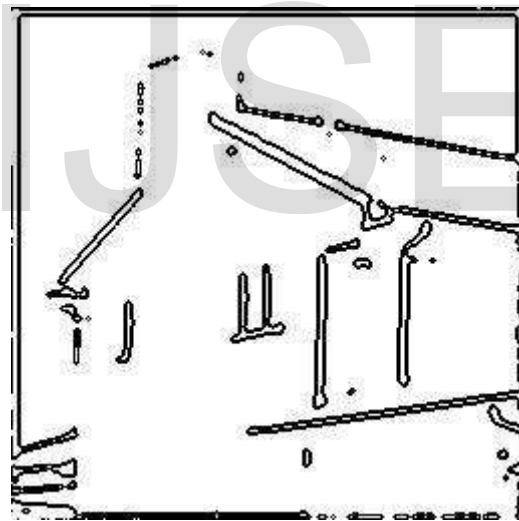


Figure 7: DOG window 13 Filtering on Original house image



Figure 8: Noisy house image

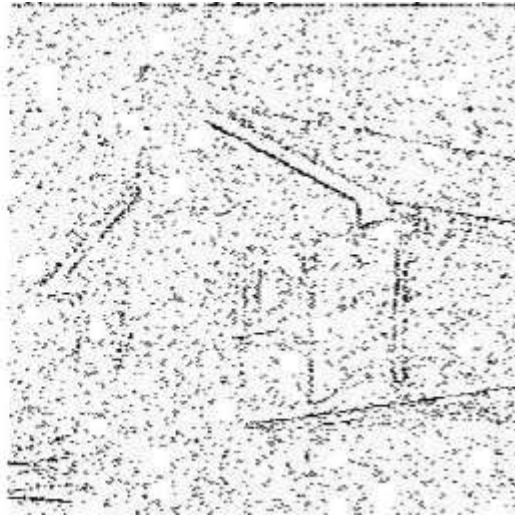


Figure 9: Roberts Filtering on Noisy house image

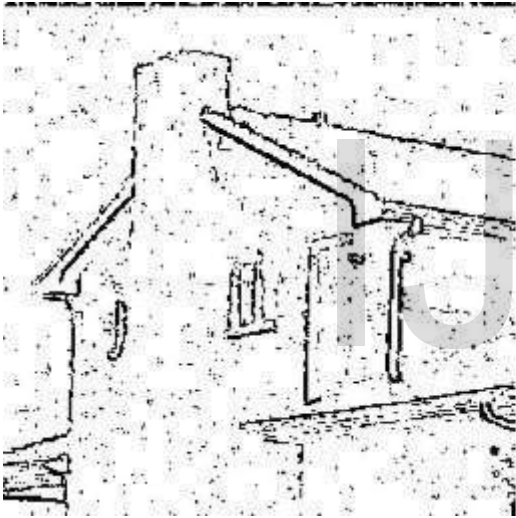


Figure 10: Prewitt Filtering on Noisy house image

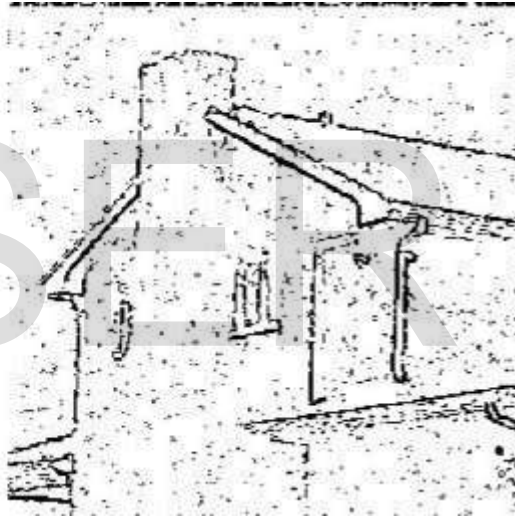


Figure 11: Sobel Filtering on Noisy house image

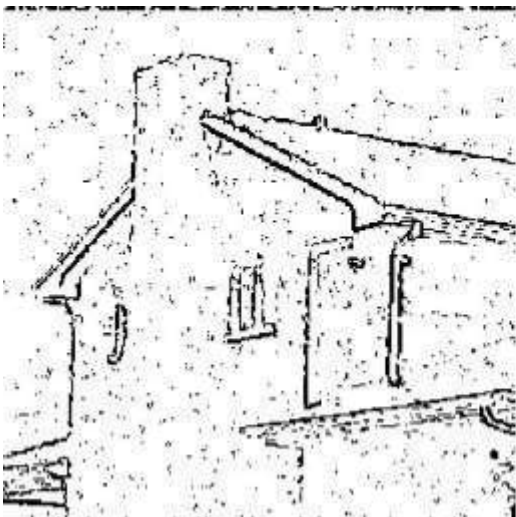


Figure 12: FreiChan Filtering on Noisy house image Figure 13: DOG window 11 Filtering on Noisy house image

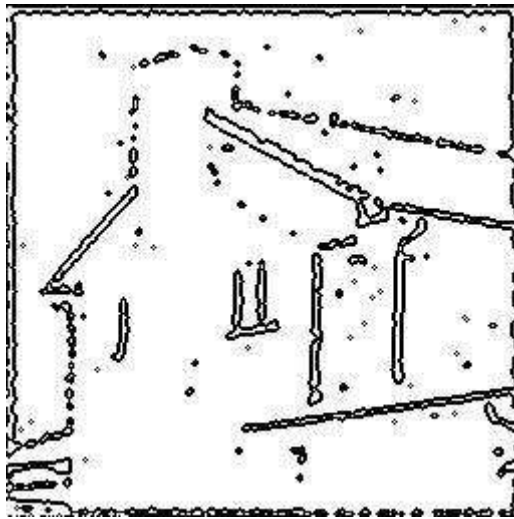


Figure 14: DOG window 13 Filtering on Noisy house image

#### IV. Discussion

The edge detection has to be performed on the figure 1 the original house image. The first order filters were applied the threshold percentage values of them was varied from 50 to 99 percentage. From among which the percentage value of 92 gave the best possible result for all first order derivatives. It can be seen from the figures 2, 3, 4 and 5 that the Roberts filter that use only two pixel values for gradient calculation performs very badly in the edge detection collecting the spurious value. The Freichan as seen in figure 5 gives the best possible result available for First order derivative. This can used to infer that if the pixel under consideration were increased and well proportioned scaling divisor applied the result can be dramatically improved

image.

From the figures 8-14 it can be seen that the noisy images give off very bad response to the edge detection, as the noise elements are too considered a high change gradient and counted into the edge detection mechanism. This creates heavy spurious values and needs to counted into edge detection by the use denoising methods. Albeit, Gaussian blurring via DOG as in figure 13, 14 does give some respite to spurious edge detection problem.

The Gaussian functions, applied by the DOG perform the very best and the change in the thresholding value by large bounds doesn't affect the output very much. The thresholding practice for the DOG seems similar to the practice of "Hysteresis". The thresholding value for the DOG was varied over the range of, 5% to 25% for the lower threshold and 80% to 95% for the higher threshold value. The optimum value for it was found to be at (10%,90%) and it was seen that the variation in the detected edges didn't show much correspondence to threshold level change. It was also noticed that with the increase in window size the filter seem to perform well as shown in figure in 6, 7 which had increased window size of 13, but at the same time by a closer inspection we can notice the marked increase in the "edge thickness" which are the side-effects of blurring the original

## Part(b): Edge Detection, Advanced Edge Detection

### I. Abstract and Motivation

Edge detection process is heavily affected by a number of parameters and the presence of noise in the input image further adds to the conundrum. Fine-tuning the parameters such as the thresholding level, the gradient map and so on depending on the image content and continual feedback is an extensive and time consuming process. The need here is to develop edge detection mechanism that can handle the variation in such parameters with high degree of stability.

Our, motivation here is to understand and devise such an edge detection mechanism. Also, the presence of noise severely affects the image detection process. So, we need to incorporate pre and post processing mechanism in the devised algorithm to take care of such anomalies effectively.

### II. Approach and Procedures

#### a) Pre-Processing

The noisy input to the edge detector produces very spurious output and affects the performance the edge detector. The best possible pre-processing mechanism for the edge detector thus can be noise removal and the most effective algorithm for this from among the ones we used, is Non-Local Mean Filtering. We apply the NLM to obtain best possible result which effectively reduces the noise and at the same time preserves the edge value.

**NLM Grayscale Filtering**, the parameters to be considered are the

d  
o  
w  
  
S  
i  
m  
i  
l  
a  
r  
i  
t  
y

w  
i  
n  
d  
o  
w

"h", filter parameter which depends on the standard deviation of the noisy image by the formula:  $h^2 = 10 \cdot SD$

The standard deviation of the noisy image was calculated by comparing with both the Mean and Weighted filter image and was found out to be:

Mean: 24.15

Weighted: 20.34

And from this the h value is averaged around 16.

With Static window size of Search window=21, Similarity window=7 which is supposed to be optimum values; h parameter was implemented by trial and error mechanism.

The h values implemented were = 16, 20, 40, 50, 44

S  
e  
a  
r  
c  
h

w  
i  
n



If Hit

HitMap[]=1

From among these  $h=44$  produced the best results which effectively removed the noise and preserved the edges for most part and was the chosen candidate.

### b) Post Processing

The edges so obtained in the edge detection are blocky and patchy, which can be improved by changing the threshold but doing it so often also affects the amount correct edges detected. So, we go for post processing mechanism which effectively rules out all such an hurdle by thinning out the edges to the minute extent.

**Thinning** is the process which tends to capture the dimension of the given shape or pattern. For example, a solid pattern like rectangle would produce a single pixel line depicting its width and convoluted patterns like hollow disc, a single pixel ring.

The given image was thinned iteratively till the convergence was reached at which point almost all the captured detail were of minimal value. The basic steps of the process are:

1) For every image point with a pixel present its corresponding bond value is calculated and stored. The bond value for both the 4-connectivity and 8-connectivity is taken into account. The bond value for 4-connectivity is 2 and for 8-connectivity is 1.

For all the image points

    If ImageData[]=1

        If it is 4-connectivity

            Bond=bond+2

        If it is 8-connectivity

            Bond=bond+1

2) Then for each of the image point with a pixel present conditional hit mask depending on the bond value of so calculated for it is applied and a hit map for the image is generated. This is pass1 which is called Conditional marking. It tends to separate out the outlying pixels which are in need for removal.

For all the image points

    If ImageData[]=1

        Apply HitMask depending on the  
        Bond value

The pattern for this is applied from the given set of pattern corresponding to the pixel bond value. The bond value under consideration for thinning process is from 4 till 10.

3) For the HitMap so generated, it is scanned and for each positive hit value unconditional mask is applied and if it is a miss the original image pixel is copied to the new image or if it

is a hit 0 is copied into the new image. This is pass 2 called Unconditional marking. It tends to find the pixels which can actually be removed with no drastic ramifications on the process.

For all the HitMap values

If HitMap[]=1

A

P

P

l

y

U

n

c

o

n

d

i

t

i

o

n

a

l

M

a

s

k

I

The patterns so matched are:

1. Spur
2. Single 4-Connection
3. 4-Connected Offset
4. Spur Corner Cluster
5. Corner Cluster
6. Tee Branch
7. Vee Branch
8. Diagonal Branch
9. L Cluster

4) The new image is then retransferred and the process is iterated till the convergence is reached.

Thus we get edges that are very fine and highly resolvable. Thinning gave the best result for edge detection.

### III. Experimental Results

Shown below are the experimental results for the problem 1(b):



Figure 1: Original noisy house image



Figure 3: NLM filtered image, h20

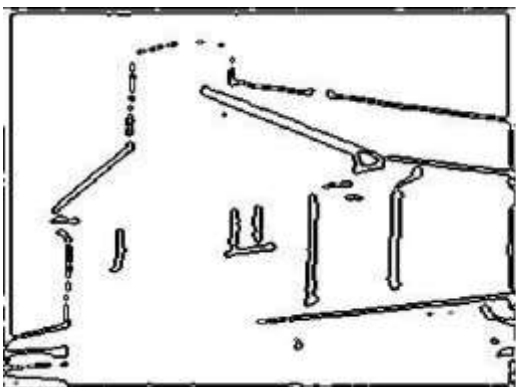
Figure 5: NLM filtered image, h50  
DOG Edge Detection from NLM filtered image

Figure 9: Thinned image of Prewitt Edge Detected Image

Figure 2: NLM filtered image, h16



Figure 4: NLM filtered image, h40



Figure 6: NLM filtered image, h44, Best Output

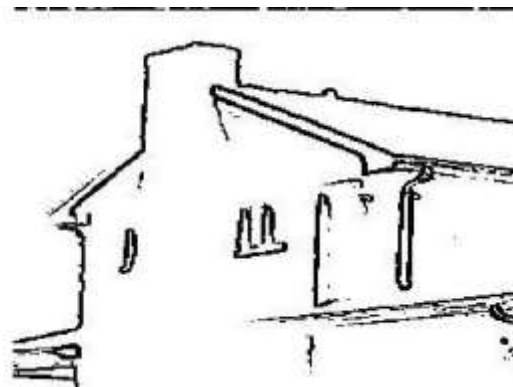


Figure 10: Thinned image of DOG Edge Detected image

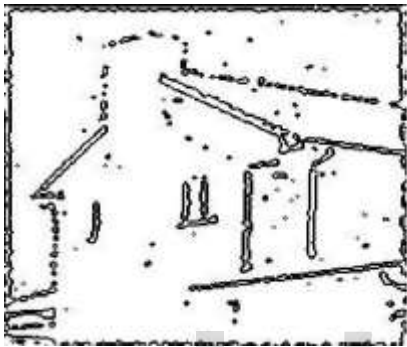
#### IV. Discussion

The figure 1 shows the noisy image of the house, which needs to undergo the edge detection process. Due to the presence of the noise component the edge detection will be severely hampered.

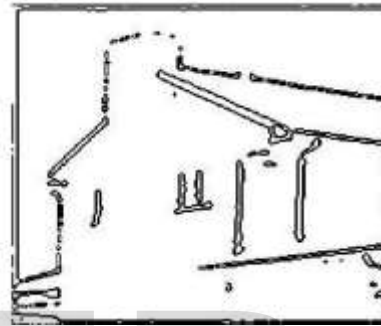
To reduce this effect, we opt for pre-processing technique that involves noise elimination using Non-Local Mean filtering. The filter was applied to

the standard search window size of 21 and similarity window size of 7 with the  $h$  value varied over the range, analysing which gave the  $h=44$  as shown in figure 6 as the best result which preserved the edges as well as reduced the noise to maximal extent. This noise filtered image was then edge detected using both Prewitt and DOG operator, as can be seen in figures 7 and 8. The comparison to the non-filter image are shown below, which clearly shows the improved performance

DOG:



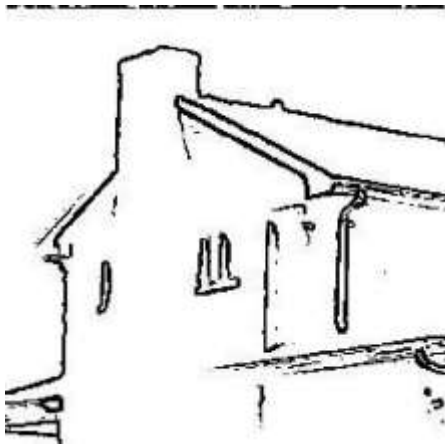
Non-filtered



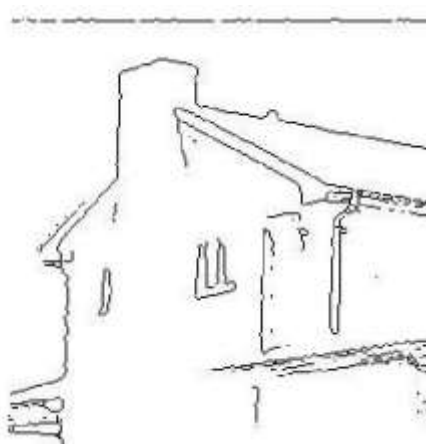
Filtered

The edges so obtained were still patchy and blocky looking, so the intuition suggested that running thinning process to make the edges fine would really help the output. The thinning process was iteratively run for both Prewitt and DOG, as seen in the figures 9 and 10. The comparison from non thinned image and thinned image gives clear perspective that the thinned image gives far better result as shown below.

Prewitt:



Non-Thinned Filtered



Thinned Filtered

The comparative result shows that implementation of pre and post processing mechanism truly improves the performance of the edge detection mechanism.



**Part(c): Edge Detection, Canny Edge Detection**

$2 * \text{ceil}(2.5 * \text{sigma})$  in the Gaussian kernel generation phase.

**I. Abstract and Motivation**

The edge detection techniques that we have studied till now relied on the noticeable variation of the intensity of the image gradient map without considering the orientation and the continuity of the edged so formed and thus hampered the process of effective edge detection. This algorithm fail in the three aspects defined for good edge detection techniques, which are 1) good detection which defines the finding the real edges correlating to the edges on the given image, 2) good localization, which defined the position of the found edge to be highly correlative to the original image and 3) minimal response, which defines that given edge information need to be specified only once.

The Gaussian smoothing is carried out separately for rows and columns, where the image pixel value is dot produced with Gaussian kernel value. The Gaussian kernel is generated at a separate function with it being called each time for both row and column smoothing.

The kernel is created from the formula:  $G(x,y) = 1/\sqrt{2\pi}S^2 \exp(-1/2(x/s)^2)$

So to encompass all these said characteristics, an algorithm needs to be developed which can effectively and efficiently find the edges using the complete set of intensity and angular value of the image gradient map. The canny edge detection technique proves to be one such an algorithm. Our motivation here is to understand the canny edge detection process and fine-tune the given parameters of the algorithm to get better results.

**II. Approach and Procedures**

The canny edge detector basically works in the following steps:

**1) Noise Reduction**

The given image is Gaussian smoothed to denoise the image and creates a slight blurred image as the output. The amount of Gaussian smoothing depends on the window size with consequently is related to the standard deviation and thus it forms the backbone of the filtering operation. If we need to trace out fine edges the Gaussian blurring has to be least and so the window size the smallest and vice versa.

*Program Explanation:*

The Canny function calls the Gaussian smoothing function first and pass on the arguments of image, standard deviation is passed along to the Gaussian Smoothing Function.

The initial standard deviation is considered to be 1 and the window size is calculated as,  $\text{Window} = 1 +$

## 2) Finding Gradient Map

After the Gaussian blurring to the extent needed is done. The gradient of the image is found by applying any four of the given filters to detect edges in all vertical, horizontal and diagonal direction. This nothing but the first order derivative of the row ( $G_x$ ) and column ( $G_y$ ) element and also, the angular gradient is found out which is approximated to (0,45,90,135) degrees to show the variation in the vertical, horizontal and two diagonal directions

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$

### *Program Explanation:*

The gradient is so found out in the program by means of calling function `derrivative_x_y()`. Which applies the linear first order derivative in the row and column values with the differential filter of (-1, 0, 1) applied both in the vertical and horizontal elements and the process being continually iterated. After this function is executed the Gradient magnitude is calculated using the formula stated above and stored by calling the function `magnitude_x_y()`.

## 3) Non-maximum suppression

This step tends to form a basic idea of the types of edges available in the image and gives binary output which traces the thin edges of the image. It looks at the gradient angle of the supposed edge point and places it along one of the four values of 0, 90, 45, 135 which would accordingly constitute as an edge when the gradient magnitude at that point is greater than the magnitude in the points at east-west, north-south, north east and south west, and north west and south east directions respectively. The thin edges are then found and stored.

### *Program Explanation:*

The function iterates for all the passed number of rows and columns and the conditional check of the current gradient magnitude being greater than ones at its side is applied for each row and column element separately (signifying the directional check) and the magnitude when turns out be larger than zero in any instance the result is noted as a “

Possible Edge” and “No Edge” otherwise. The value of the Possible edge and No Edge was specified to be 128 and 0 in the program definition.

## 4) Hysteresis Thresholding

These no hard and fast rule to provide the thresholding levels for the images as the intensity levels corresponding to images are not easy to define and applied in a ad-hoc manner, it is

tailored to each image separately and uniquely. The chances are that depending on the threshold we may lose the edges if it's higher and gain extraneous noise if it's too low. So, we use two level thresholding called hysteresis, wherein the intensity points above the high threshold are for sure considered to an edge point and the intensity points below the low threshold level are considered to non-edge point. And by the use of information from the non-maximum suppression and following edge tracing we can best determine the edges of the image in a minimalistic, localized manner.

#### *Program Explanation:*

The two levels of the thresholding is found first iterating the function for the pixels which are considered to be the "Possible Edge". And for each such possible edge values the histogram value is devised by means of iteratively counting the number of pixel values with given intensity values and incrementally checking it for next intensity value. Also, a "highcount" value is found which defines the maximum bound for the count process and is derived from the count of the number of pixels that passed the non-maximal suppression. The high threshold is then the percentage count value of number of edges till the highcount. The lower s noted in the output image.

threshold is then given as the divisor factor scaled value of highthreshold.

$$\text{Lowthreshold} = \text{Highthreshold} * \text{tlow} + 0.5$$

#### 5) Edge Tracing

With the application of the HighThreshold, we can find out the definite presence of the outline of the edge and with the help of the directional information we can scourge out the neighbouring pixel point which qualify the lower threshold and inductively find the entire edge.

#### *Program Explanation:*

The follow\_edges() function performs this function which compares the neighbouring set of causal pixel for both row and column values with mask pattern of {1,1,0,-1,-1,-1,0,1} for row and {0,1,1,1,0,-1,-1,-1} for column to define the definite possibility of probable edge to be a edge, when the value of the so masked vector is greater than the defined lowvalue and the pixel under consideration is a Possible Edge.

For the given code, the value of the lower and upper threshold was varied along with the sigma value for both noisy and original image and the variation wa

### III. Experimental Results

Given here are the experimental results for the problem 1(c):



Figure 1: Original house image



Figure 2: Canny edge detected original house for sigma=1, LowT=70 % and HighT=85%

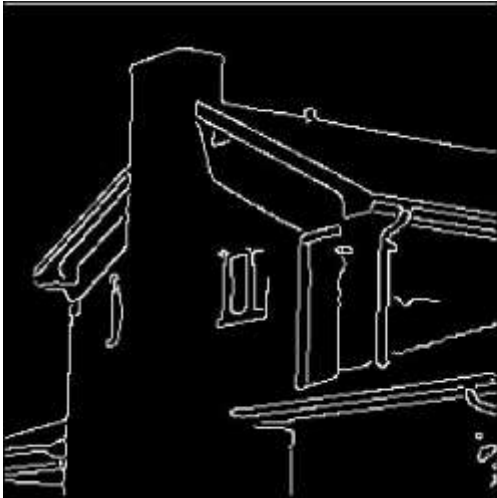


Figure 3: Canny edge detected original house for sigma=1, LowT=60 % and HighT=85%



Figure 4: Canny edge detected original house for sigma=1, LowT=60 % and HighT=95%



Figure 5: Canny edge detected original house for sigma=1.5, LowT=70 % and HighT=85%

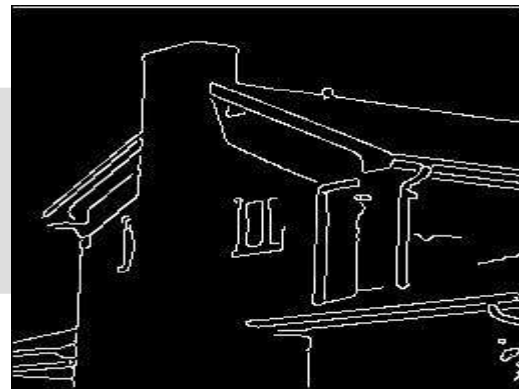


Figure 6: Canny edge detected original house for sigma=0.5, LowT=70 % and HighT=85%

#### IV. Discussion

It can be noticed that the canny edge detection mechanism clearly improves the process of edge detection. Comparing the results obtained from First-order derivative, second order derivative and canny edge detection for noisy house image:



We can clearly see that the canny edge detection performs way better in terms of the edge localization, detection and minimal response. It gains the advantage of using the angular gradient detail among others.

The size of the Gaussian filter used which depends on the Standard deviation value directly affects the output result of the Canny Edge detection. As, the smaller filter cause less blurring which allows the detection of small, sharp lines. This can be seen in the figures 6 and 12 with sigma value of 0.5, which produces minute edge details. And likewise, larger the filter more the blurring and the prominent edges only show up. This can be seen in figure 5 and 11 which has the sigma value of 1.5, which produces the edge only for the larger, more prominent values. Also, with the decrease in the lower threshold value more edge details are seen, as shown in the figures 3 and 9. This is so because the number of "possible edge" points considered by the algorithm increase by large bounds. And similarly, with the increase in the upper threshold value

less and only prominent edge details are seen, as shown in the figures 4 and 10. This is so because the number of “exact edge” points considered by the algorithm decreases and during the process of edge tracing the information from some of the “possible edge values is lost”.

So it is seen that we can tailor the canny edge detector in a simple manner to suit any kind of processing need.

## MORPHOLOGICAL PROCESS

### Part(a): Morphological Processing, General Morphological Operations

#### I. Abstract and Motivation

Morphology relates to the study of a structure and morphological processing for images means digital image processing techniques on the shapes and forms of the image. Morphological processing forms a key element in area where the size or texture of image or a pattern within it doesn't matter; all the information needed at that juncture is the shape and general outline of the object. The example of such situations is counting the number of cars passing through a traffic light from the image obtained from the surveillance camera. The form of cars are recognised and the counting of the car forms is done on time scale.

The same technique can be applied to varied fields from counting the eyeball hit for an commercial to the analysis of electronic circuitry. Our motivation in this section is to develop and fine-tune one such technique to count the number holes and pathways in the given PCB board. The same technique can also be generalised to be applied into all the said examples and can incorporate more solutions. The motivation here is also to use the complete set of basic morphological processing tools available and develop the technique which is efficient, effective and can be broadly be generalised. The output of such a processing depends on many parameters, which are furthermore dependent on the assumptions made on the part of the contents of the image. So, further we have to develop intuition to make such assumptions that stays valid for varied circumstances.

deal with the problem of counting the number of holes and pathways. Albeit the image and the detail fusion given here can't just be solved by morphological processing and there's dire need to develop image adaptive techniques to find the number of holes and pathways.

The preliminary steps would be to get the simplest and most complete outline of the given PCB board image that would help in getting the needed details with the least amount of effort and good effectiveness. The set of morphological process for such requirements are:

- a) Shrinking
- b) Thinning
- c) Skeletonising

#### II. Approach and Procedures

There are many approaches to this problem and it all comes down to the two factors: level of simplicity and efficiency; and effectiveness of the technique. If we can have a really good morphological processing technique and a wholesome image to deal with, we can have simplest of algorithms to

**Skeletonising**, it is the process which tends to capture the angular detail of the given image in the best possible minimal way. For the process of finding holes and pathways the need of angular details of the image is void and there's no solid information to be retrieved from such an image. So, we pass over using Skeletonising.

**Shrinking**, it is the process which tends to get to the basic definable shape of the given pattern. For example for a solid pattern the base pattern would be a single pixel and for convoluted patterns like a disc with hole it would be single pixel ring. This process could be used for finding holes but it erases certain set of lines which are part of certain pathways. This can be seen in the figure 3 of the result. So, Shrinking is also a no go considering the completeness of the image in concern.

**Thinning**, it is the process which tends to capture the dimension of the given shape or pattern. For example, a solid pattern like rectangle would produce a single pixel line depicting its width and convoluted patterns like hollow disc, a single pixel ring. The thinned image for the PCB thus provides the completeness in the information provided both the holes and pathways in the least amount of details with holes being represented as single pixel ring, pathways as single pixel line and almost every other extraneous details removed. This can be seen in the figure 4 of the result. So, we take thinning as the base processing technique and carry on from there on.

To make the image more hole or pathway differentiable, some more techniques can be tried out like:

**Interior Fill** first and **Shrinking** after that would make the hole complete first and then during shrinking reduce them as a part of the line. But, it was later found out that finding holes becomes integral part of finding the Pathways, so this technique had to be dropped. This can be seen in figure 5 of the result.

**Line Breaking** first and **Shrinking or Thinning** to remove out the line patterns and have only holes in the image. But, as explained earlier finding holes and pathways could be integrated into one simple step and hence it became a moot point to eliminate one or the other. This can be seen in the figure 6 of result.

So, **Thinning** forms the base morphological processing technique here and the whole count algorithm based on it.

Linear row-wise and column-wise scanning of the image gave out specific details on the characteristics of hole and pathways. This were used to make basic assumptions during algorithm implementation and after minor finetuning provided good results.

1) The hole were found to be 7 pixel wide at the maximum, which gives us the parameter to separate out spurious count values by the detection of the spaces between any two holes, pathways or both.

2) The hole was found to be almost equi-sized in both vertical and horizontal directions, giving us a pointer for elimination of spurious count value and apply the above assumption on the vertical scan also.

3) The pathways starts at the outer bounds of the holes, which are 7 pixel wide and thus scan for them can iterate from 3 pixel point

4) There can be only 2 pathways at the maximum emerging from a hole and the always start or end at hole. Thus giving us a parameter to distinguish pathways from other elements

1) For every image point with a pixel present its corresponding bond value is calculated and stored. The bond value for both the 4-connectivity and 8-connectivity is taken into account.

**Note:** the Pathway finding technique implemented here is based on the Professor Kuo's definition of Pathways in the class, which is defined as the single connected line between two holes and each of such pathways are considered separate and independent, even if they diverge out of a node.

The process and the steps so followed are:

**First**, convert the given RGB colour image to binary image. This is done in two steps of conversion of RGB colour image to grayscale and then by basic fixed thresholding technique conversion of grayscale to binary image.

RGB to Grayscale conversion:  

$$I = (0.299 * R) + (0.587 * G) + (0.114 * B)$$

Where,

I – Intensity value

R – Red component of the colour image

G – Green component of the colour image

B – Blue component of the colour image

Grayscale to Binary conversion:

For all image pixels

If Imagevalue[] > Threshold

BinaryImagevalue[] = 1

Else

BinaryImagevalue[] = 0

The image is then stored. The threshold value was varied from 10 to 240 which results varying from blocky images to no detailed images and then the threshold value of 65 was found to be optimum.

**Second**, the given image was thinned iteratively till the convergence was reached at which point almost all the captured detail were of minimal value. The basic steps of the process are:



```

For all the image points
  If ImageData[]=1
    If it is 4-connectivity
      Bond=bond+2
    If it is 8-connectivity
      Bond=bond+1

```

2) Then for each of the image point with a pixel present conditional hit mask depending on the bond value of so calculated for it is applied and a hit map for the image is generated. This is pass1

```

For all the image points
  If ImageData[]=1
    Apply HitMask depending on the
    Bond value
      If Hit
        HitMap[]=1

```

3) For the HitMap so generated, it is scanned and for each positive hit value unconditional mask is applied and if it is a miss the original image pixel is copied to the new image or if it is a hit 0 is copied into the new image

```

For all the HitMap values
  If HitMap[]=1
    A
      p
      p
      l
      y
      U
      n
      c
      o
      n
      d
      i
      t
      i
      o
      n
      a
      l
      M
      a
      s
      k

```

```

I
f
H
i
t
      NewImageData[
      ]=0
Else
      NewImageData[
      ]=
      OriginalImageD
      ata[]

```

4) The new image is then retransferred and the process is iterated till the convergence is reached.

**Third**, the thinned image is then used to first find the number of holes. The process to find the number of holes is twofold; this is so done to eliminate spurious reading due to some other elements present. The basic intuition used here is that while scanning through the image row-wise a hole would make itself present by the sudden variation from black pixel to white and then black again within small window of reference. Now this intuition coupled with the knowledge of the image and the assumptions made above that the holes are to be considered at max of 7 pixel wide and they equ-sized in every direction would helps us determine the effective algorithm. The scanning is done row-wise which when gives out the hole found read is then checked column-wise in a narrow band to confirm the location such an hole. The narrow band of search is again based on the assumption of the hole width to be of 7 pixel at max and thus producing a window band of 11 pixel high and adaptively determined width for the vertical search. This inclusion of the assumption improves the performance and efficacy of the algorithm dramatically. The process flows as following:

1) The complete image is searched row-wise for the hole hit trigger.

a) The initial scan is for finding the presence of any image element altogether. If it is so the next step of finding hollow space follows and on. If any element is found a flag is set to continue down to next action and also, the current column value is stored for the both calculation of the separation width in the final stage to determine if it is a hole or not and if so, to have to first column bound in the search window for vertical search.

For all image pixels

I

f

I

m

a

g

e

D

a

t

a

[

]

=

1

i

f

l

a

g

1

=

1

ColumnStart=CurrentColumn

b) If any image element is so found, the flag 1 is set and the algorithm goes on searching for hollow part of the hole in the near neighbourhood which is the zero part. If it so, the second flag is set and next step is followed.

For all image pixels

I

f

I

m

a

g

e

D

a

t

a

[

]

=

0

a

n

d

I

f

i

f

l

a

g

1

=

1

i

f

l

a

g

2

=

1

c) If the hollow element is found and the second flag is set, the algorithm now searches for image element which is 1 so that it can be inferred as the closing point of hole. It is found then flag 3 set, the process move on to narrow vertical search, all the other flags are reset and the current column value is store for the upperbound of the search window and then width of the suspected hole is calculated

For all image pixels

I

f  
I  
m  
a  
g  
e  
D  
a  
t  
a  
[  
]  
=  
0  
a  
n  
d  
I  
f  
i  
f  
l  
a  
g  
2  
=  
1  
i  
f  
l  
a  
g  
3  
=  
1  
i  
f  
l  
a  
g  
2  
=  
i  
f  
l  
a  
g  
1

=  
0  
C  
o  
l  
u  
m  
n  
E  
n  
d  
=  
C  
u  
r  
r  
e  
n  
t  
C  
o  
l  
u  
m  
n  
h  
o  
l  
e  
w  
i  
d  
t  
h  
=  
C  
o  
l  
u  
m  
n  
E  
n  
d  
-  
C  
o

l  
u  
m  
n  
S  
t  
a  
r  
t

2) The vertical search is carried on only is both the flag 3 is set and hole width is within 7 pixel range. The vertical search follows the same process but the search range is between Column Start and End range values and variation of  $\pm 5$  for the current row value.

a) Initial image element search

For image pixels lying in window  $\pm 5$  of row value and Column start & end values

If ImageData[]=1  
jflag1=1

b) Hollow space search

For image pixels lying in window  $\pm 5$  of row value and Column start & end values

If ImageData[]=0 and If jflag1=1  
jflag2=1

a) Final image element search. When it is confirmed the flag3 is set and the counter for hole increments in value.

For image pixels lying in window  $\pm 5$  of row value and Column start & end values  
 If ImageData[]=1 and iflag2=1  
     jflag3=1  
     HoleCounter=HoleCounter+1

Thus, the hole is found unambiguously for the given PCB image

index can be the 4 pixels down the first hit in the hole determination, because the hole is considered to be at max 7 pixel wide with midvalue is at ceiling of 4 and the first hit of the hole determination is a the top point of the hole.

So,      ColumnMidValue = ColumnStart + ColumnEnd/2;  
         RowMidValue = CurrentHoleRowValue+4

The circular search then starts from these values,

For 11 iteration  
     If ImageData[CircularIndex]=1

**Fourth**, the process to find line starts as soon as presence of hole determined. The process of finding holes comes down relying on the last two assumption made above which are 1) The pathways starts at the outer bounds of the holes, which are 7 pixel wide and thus scan for them can iterate from 3 pixel point and 2) There can be only 2 pathways at the maximum emerging from a hole and the always start or end at hole. Thus giving us a parameter to distinguish pathways from other elements.

We base this knowledge and assumption to create the algorithm to count the number of pathways which is made of two concatenated step, the first step scan the neighbourhood of the hole from its midpoint circularly in a incremental fashion for certain number of iterations, finds the number of elements hit in each search and stores their index values. The second step carries one only when the number of elements found in the circular search is one, which amounts to finding a part of the line and then finds out if the same pattern is repeated for next two consecutive searches which amounts to an actual line segment. This is then confirmed by checking the difference between the row and column index of the consecutive hit, which will show to be a line only when the difference in less than two at max accounting for line shift in both vertical and horizontal direction.

The basic process steps can be put down as:

1) Circularly searching the area around hole from its mid-point in an incremental manner to find if any pathways emerges form it in the form of a line. There are couple of parameters to be taken care of here, the midpoint can be derived for column point as an average of the Column Start and End values found in the hole search process and the row

```

NumberofElements=
NumberofElements+1
NewRowIndex=CurrentRowIndex
NewColumnIndex=CurrentColumnIndex

```

The difference of the index value is calculated to effectively determine the line confirmation.

```

IndexDifference = |NewColumnIndex-
OldColumnIndex| + |NewRowIndex-OldRowIndex|

```

2) For each element hit found, when the number elements so found in the circular search is exactly equal to one, the difference is compared and the process is done for two more steps. Consequently determining the Pathway found.

a) If there's an element found and the number of elements so found is exactly 1 the flag1 is set and the process moves onto next step

```

If NumberofElements=1
    pflag=1

```

b) If there's an element found again, the number of elements so found is exactly 1 and the flag1 is set, the flag is

incremented and the process moves onto next step.

```

If NumberofElements=1
    pflag=2

```

c) If there's an element found again, the number of elements so found is exactly 1 and the flag1 is 2, a suspected line or pathway is then found, the flag is reset and the process moves onto next step

```

If NumberofElements=1
    pflag=0

```

d) If for the suspected pathway the difference in index is at max two at both the above stage b,c the pathway is then definitely found and the pathways counter is incremented.

For above two steps

```

If IndexDifference<=2
    PathwayCounter++

```

The point to be noted here is that even if a hole has two Pathways connected to it, it'll determine only one and the next one will be determined by the next connected hole to the pathway till the last pathway is found by last connected hole thereby producing valid results.

### III. Experimental Results

Shown below are the results for the problem 2(a):

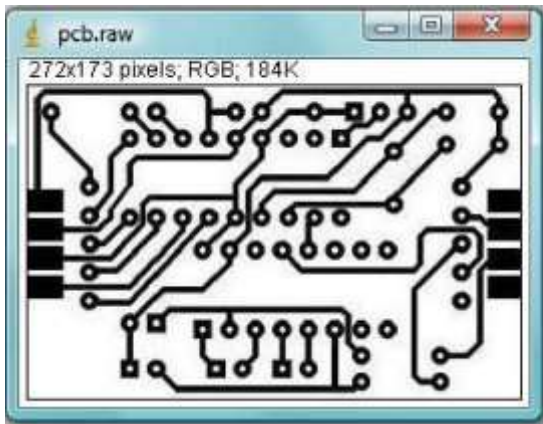


Figure 1: Original RGB PCB image

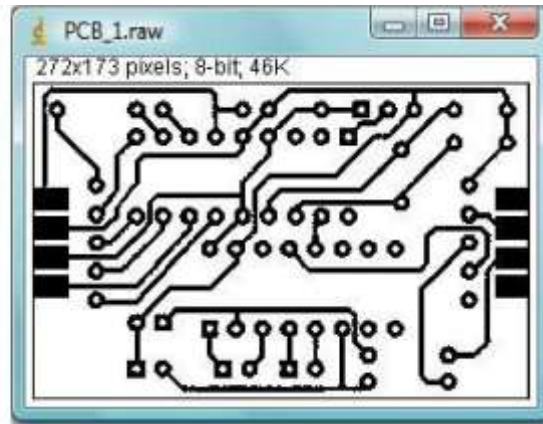


Figure 2: Bitmap converted PCB image

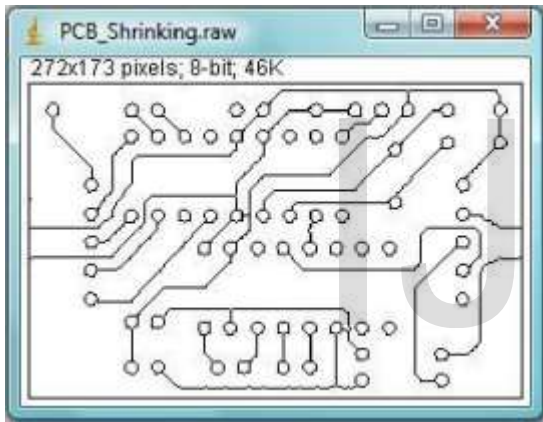


Figure 3: Shrunk PCB image

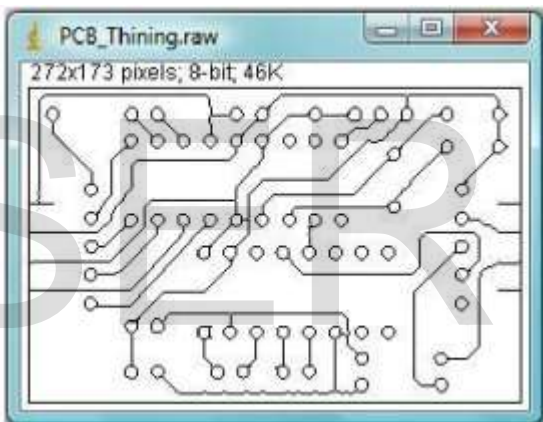


Figure 4: Thinned PCB image

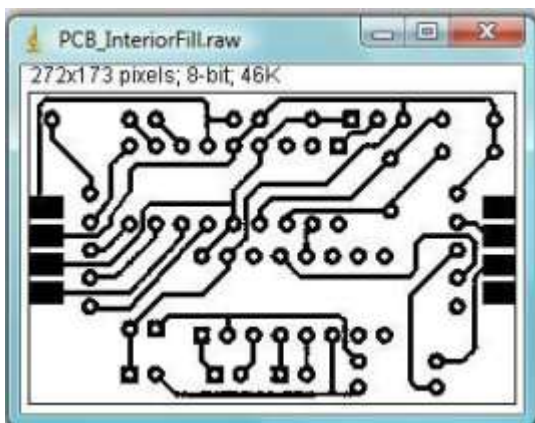


Figure 5: Interior Filled PCB image

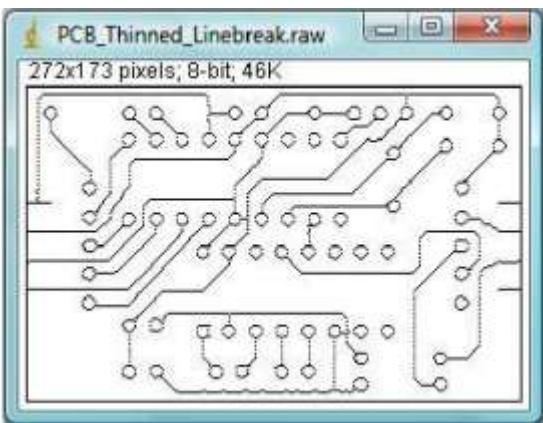


Figure 6: Line breaking on thinned PCB image



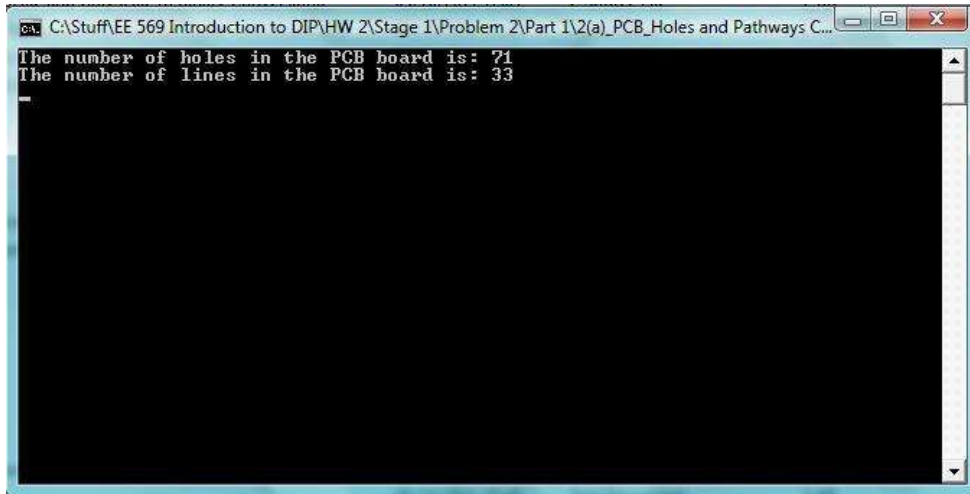


Figure 7: Final Output - Number of Holes and Pathways

#### IV. Discussion

The determination of the holes and pathways for given PCB image depends on a number of factor such as the choice the parameters, the assumptions made and the way the algorithm is implemented.

For the given algorithm the main tenet upon which all the assumptions were made was the supposed size of the hole which after analyzing the given image and careful finetuning was found to be 7 at the max. When the size of the hole was assumed to vary between 5 to 9 and all the corresponding changes in the algorithm parameters like the row midpoint index, search window and on was made following values of the holes and pathways was observed:

Size: 9

Holes: 48

Pathways: 19

Size: 5

Holes: 54

Pathways: 21

Size: 6

Holes: 69

Pathways: 27

Size: 7

Holes: 71

Pathways: 33

Size: 8

Holes: 63

Pathways: 23

It can be seen that the values drop off linear after the peak at the size = 7 and seem to converge at point of size = 7. This validates our assumptions and algorithmic procedure.

The quality of the input image also plays a pivotal role here. For comparison of the effect of quality on the efficacy of the algorithm four binary images were created from the grayscale PCB image applying varying levels of threshold of 50, 65(optimum level), 150 and 200 respectively. The binary image was then appropriately thinned and processed. The algorithm was applied and the values were obtained as:

Threshold: 50  
Holes: 72  
Pathways: 33

Threshold: 65  
Holes: 71  
Pathways: 33

Threshold: 150  
Holes: 68  
Pathways: 29

Threshold: 200  
Holes: 65  
Pathways: 26

IJSER

As we know the with the increased threshold, more of the image content was lost in the conversion to binary image which when subsequently thinned eliminated few of the image components and thus the hole and pathway count also drops off. The decreased threshold creates an opposite effect and we seem to notice a minute increase in the number of holes and pathways so found.

So, from the above discussion it can be clearly inferred that along with good algorithm we need to have better fine-tuned parameters to effectively find the count values and get the desired results.

The algorithm's efficacy is further hampered due to the fact that the separation between hole and certain elements is of the same width as the hole and the result so produced was spurious. But having the double scanning check and tightly fine-tuning the parameters gave the best possible result.

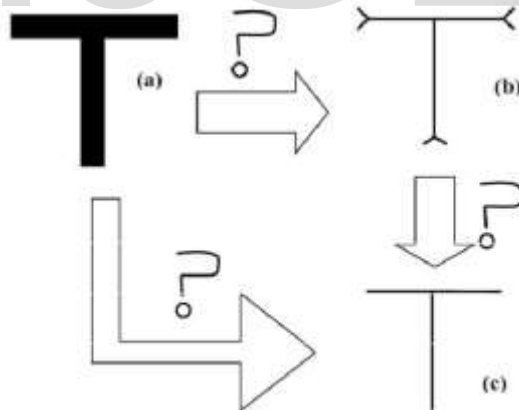
## Part(b): Morphological Processing, Thinning and Skeletonising

### I. Abstract and Motivation

It becomes imperative in certain situation that we use just the parametric data of the image such as its outline, dimension and connectivity without giving much heed to the image as such. We use morphological processing for such a situation like optical character recognition or pattern detection. The morphological process acts on the shape or the form of the image, rendering the information need in a minimal form in the output.

Our motivation here is to study two such main forms of morphological processing, thinning and skeletonising. We are to understand it functioning and analyse it in a step-wise manner. Also, there's the need to study the effect of running them consequently and devise a better way for the transformation between their forms.

### II. Approach and Procedures



seen in the image (b) that it defines the dimensions and the edge connectivity of the original T image in the form of connected line and continual spurred lines.

*(ii) What morphological operation/process is needed for (a) to (c)? Explain.*

The process applied here is Thinning. Thinning is the process of recursively removing the foreground region to reduce the given image to an eroded remain which defines the

*(i) What morphological operation/process is needed for (a) to (b)? Explain.*

The process applied here is Skeletonising. Skeletonising is the process of recursively removing the foreground region to reduce the given image to a skeletal remain which defines the extent and connectivity of the original region. This is nothing but the angular detail of the given image. It can be

dimensions and only the extent of the original image. It can be seen in the image (c) that it defines the dimension and the middle bound characteristics of the original T image in the form of two perpendicular lines.

*(iii) What morphological operation/process is needed for (b) to (c)? Explain.*

The process applied is Spur removal. Spur removal is the process of recursively removing the slanted lines which are free at any of their end point. It is easily noticeable that the Skeletonised image (b) and the Thinned image (c) are similar in every manner except the part of the spurred lines. Hence, by the application of spur removal process we can derive image (c) from image (b).

**Thinning** is the process which tends to capture the dimension of the given shape or pattern. For example, a solid pattern like rectangle would produce a single pixel line depicting its width and convoluted patterns like hollow disc, a single pixel ring.

The given image was thinned iteratively till the convergence was reached at which point almost all the captured detail were of minimal value. The basic steps of the process are:

1) For every image point with a pixel present its corresponding bond value is calculated and stored. The bond value for both the 4-connectivity and 8-connectivity is taken into account. The bond value for 4-connectivity is 2 and for 8-connectivity is 1.

```

For all the image points
  If ImageData[]=1
    If it is 4-connectivity
      Bond=bond+2
    If it is 8-connectivity
      Bond=bond+1

```

2) Then for each of the image point with a pixel present conditional hit mask depending on the bond value of so calculated for it is applied and a hit map for the image is generated. This is pass1 which is called Conditional marking. It tends to separate out the outlying pixels which

are in need for removal.

```

For all the image points
  If ImageData[]=1
    Apply HitMask depending on the
    Bond value
    If Hit
      HitMap[]=1

```

The pattern for this is applied from the given set of pattern corresponding to the pixel bond value. The bond value under consideration for thinning process is from 4 till 10.

3) For the HitMap so generated, it is scanned and for each positive hit value unconditional mask is applied and if it is a miss the original image pixel is copied to the new image or if it is a hit 0 is copied into the new image. This is pass 2 called Unconditional marking. It tends to find the pixels which can actually be removed with no drastic ramifications on the process.

For all the HitMap values

If HitMap[]=1

A

P

P

l

y

U

n

c

o

n

d

i

t

i

o

n

a

l

M

a

s

k

I

f

H

i

t

NewImageData[  
]=0

Else

NewImageData[  
]=  
OriginalImageD  
ata[]

10. Spur
11. Single 4-Connection
12. 4-Connected Offset
13. Spur Corner Cluster
14. Corner Cluster
15. Tee Branch
16. Vee Branch
17. Diagonal Branch
18. L Cluster

4) The new image is then retransferred and the process is iterated till the convergence is reached.

**Skeletonising** is the process of recursively removing the foreground region to reduce the given image to a skeletal remain which defines the extent and connectivity of the original region. This is nothing but the angular detail of the given image.

The given image was skeletonised iteratively till the convergence was reached at which point almost all the captured detail were of minimal value. The basic steps of the process are:

1) For every image point with a pixel present its corresponding bond value is calculated and stored. The bond value for both the 4-connectivity and 8-connectivity is taken into account. The bond value for 4-connectivity is 2 and for 8-connectivity is 1.

For all the image points

If ImageData[]=1

If it is 4-connectivity

Bond=bond+2

If it is 8-connectivity

Bond=bond+1

The patterns so matched are:

2) Then for each of the image point with a pixel present conditional hit mask depending on the bond value of so calculated for it is applied and a hit map for the image is generated. This is pass1 which is called Conditional marking. It tends to separate out the outlying pixels which are in need for removal.

For all the image points

If ImageData[]=1

Apply HitMask depending on the  
Bond value

If Hit

HitMap[]=1

The pattern for this is applied from the given set of pattern corresponding to the pixel bond value. The bond value under consideration for thinning process is from 4 till 11.

3) For the HitMap so generated, it is scanned and for each positive hit value unconditional mask is applied and if it is a miss the original image pixel is copied to the new image or if it is a hit 0 is copied into the new image. This is pass 2 called Unconditional marking. It tends to find the pixels which can actually be removed with no drastic ramifications on the process.

For all the HitMap values

If HitMap[]=1

A

P

P

l

y

U

n

c

o

n

d

i

t

i

o

n

a

l

M

a

s

k

I

f

H

i

t

NewImageData[  
]=0

Else

NewImageData[  
]=  
OriginalImageD  
ata[]

The patterns so matched are:

1. Spur
2. Single 4-Connection
3. L Corner
4. Corner Cluster
5. Tee Branch
6. Vee Branch
7. Diagonal Branch

4) The new image is then retransferred and the process is iterated till the convergence is reached.

**Spur removal** is the process of recursively removing the slanted lines which are free at any of their end point.

The given skeletonised image was spur removed iteratively till the convergence was reached at which point it resembled the thinned image. The basic steps of the process are:

- 1) For every image point with a pixel present its corresponding bond value is calculated and stored. The bond value for both the 4-connectivity and 8-connectivity is taken into account. The bond value for 4-connectivity is 2 and for 8-connectivity is 1.

```

For all the image points
  If ImageData[]=1
    If it is 4-connectivity
      Bond=bond+2
    If it is 8-connectivity
      Bond=bond+1

```

- 2) Then for each of the image point with a pixel present and the bond value was exactly 1 the flag was set, which meant the current pixel has to be considered for pattern matching.

```

For all the image points
  If ImageData[]=1 and If Bondvalue=1
    Flag1=1

```

- 3) If the flag is set the next step is to find if the pixel points around the given image match the spur removal pattern and if so, the pixel under consideration is removed and flag reset.

```

For all the image points
  If flag1=1
    If
      CurrentPixelPattern=SpurPixelPattern
      NewImageData[]=0
      Flag=0
    Else
      NewImageData[]=
      OriginalImageData[]

```

- 4) The new image is then retransferred and the process is iterated till the convergence is reached and the output image resembles the thinned image output.



### III. Experimental Results

Shown below are the results for the problem 2(b):

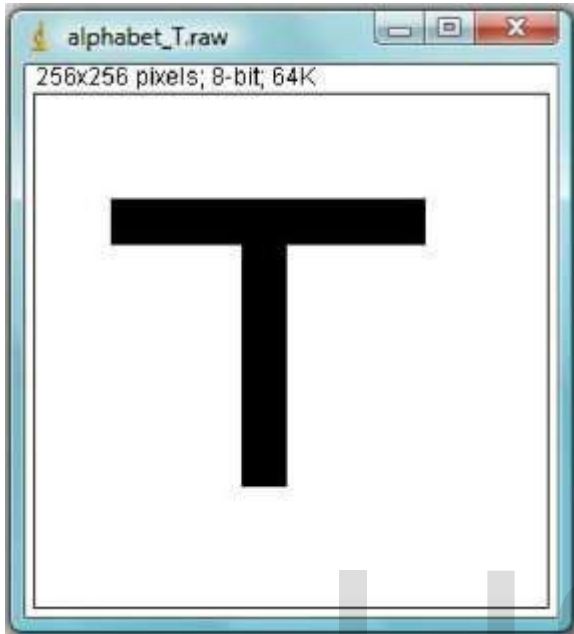


Figure 1: Original T-alphabet

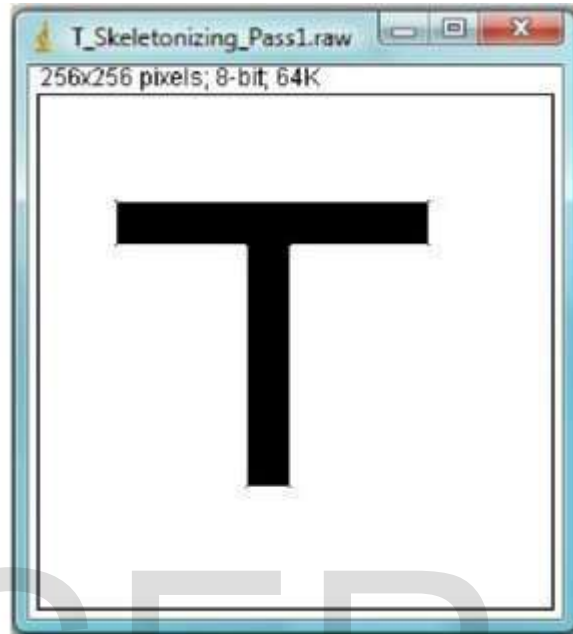


Figure 2: Skeletonized T after 1 iteration

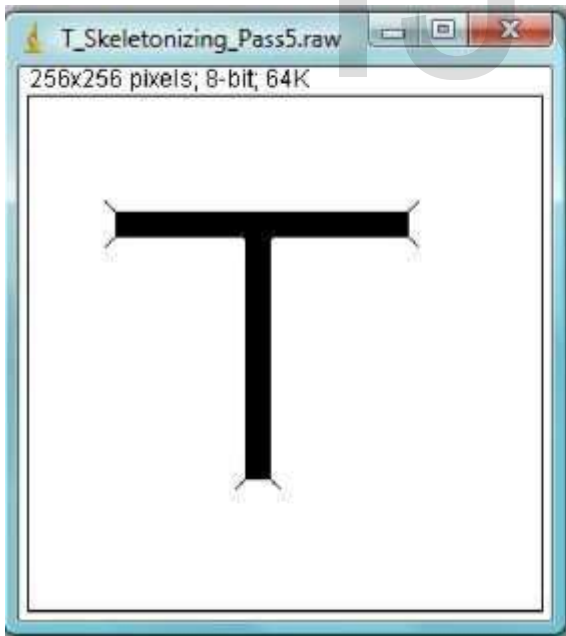


Figure 3: Skeletonized T after 5 iterations

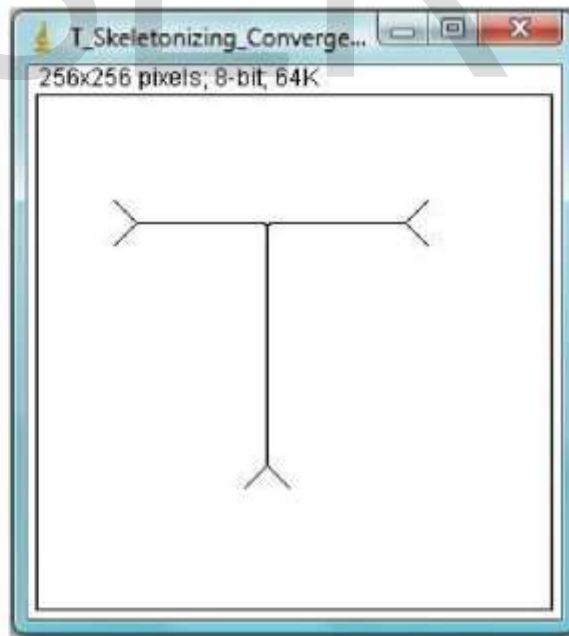


Figure 4: Skeletonized T after convergence

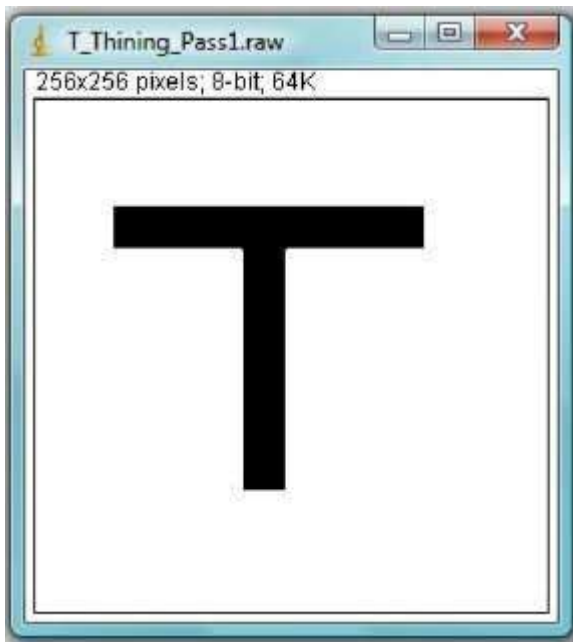


Figure 5: Thinned T after 1 iteration

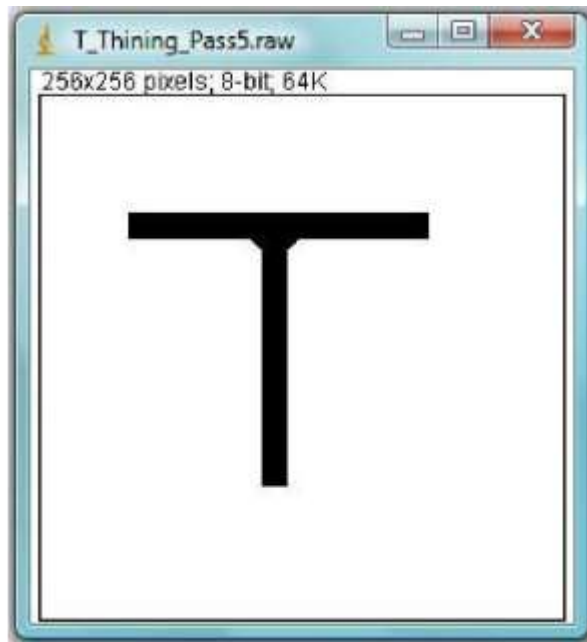


Figure 6: Thinned T after 5 iterations

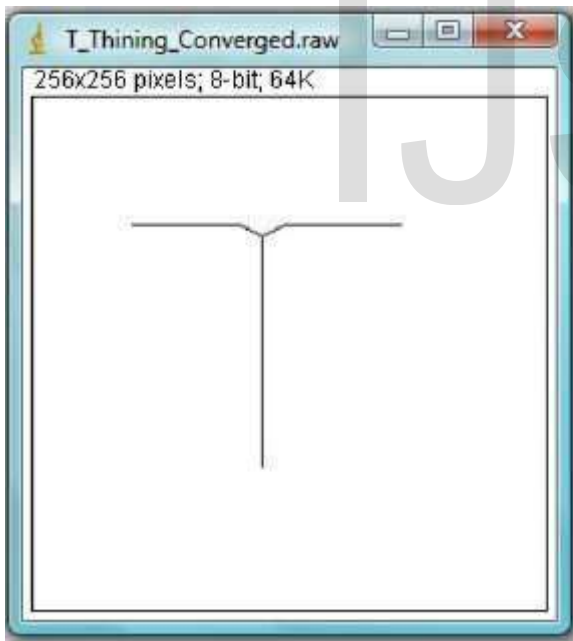


Figure 7: Thinned T after convergence

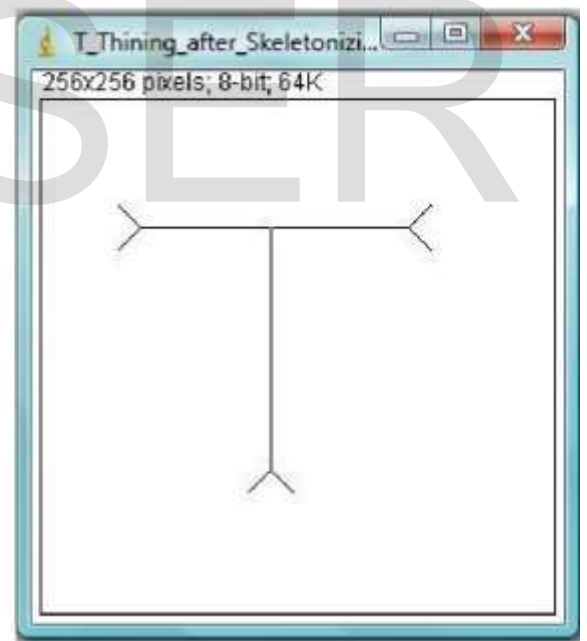


Figure 8: Thinned T after Skeletonizing

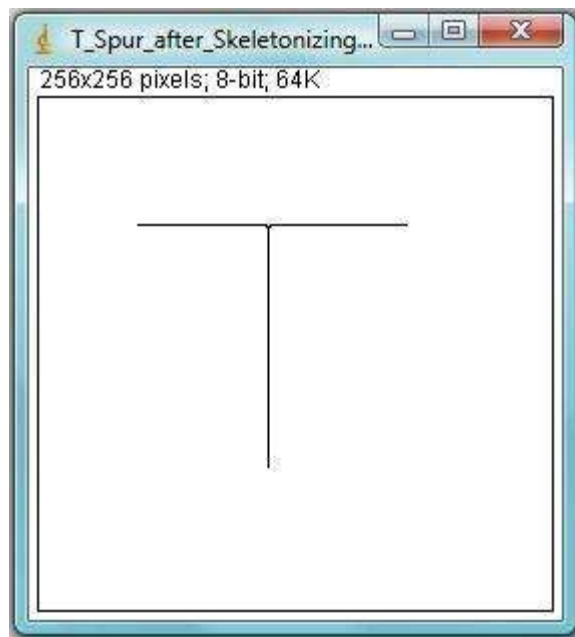


Figure 9: Spur removed T after Skeletonizing, Needed Image

#### IV. Discussion

The figure 1 shows the original T alphabet image upon which the morphological processing of skeletonising and thinning has to be done and studied. We also need to verify the efficacy of the approach by comparing the obtained result from the ones given. Also, find out if spur removal is the process for equating paths [(a)->(b)->(c)] and 2[(a)->(c)].

The figures 2, 3 and 4 illustrate the gradual process of skeletonising, after 1<sup>st</sup>, 5<sup>th</sup> and convergence respectively. It can be seen in the figure 2 that the 1<sup>st</sup> pass just removes the T image initial boundary leaving out the corner pixels intact as they represent the T image connectivity. Now further down the lane in the 5<sup>th</sup> pass it can be seen in the figure 3 the process of foreground removal excluding the corner ones being very much prominent. The convergence of the solid image like T alphabet depends on its thickness such that more thick the image component is more time it'd take for the removal of the not needed foreground. For the given image the process of skeletonising needs 12 iterations to converge as shown in figure 4.

The figures 5, 6 and 7 illustrate the gradual process of thinning, after 1<sup>st</sup>, 5<sup>th</sup> and convergence respectively. It can be seen in the figure 5 that the 1<sup>st</sup> pass just removes the T image initial boundary leaving out just one pixel each in the inner folds of T image. The 5<sup>th</sup> pass processing as in figure 6 it can be seen the process of foreground removal with dimension

semblance being very much prominent. The convergence of the solid image like T alphabet depends on its thickness such that more thick the image component is more time it'd take for the removal of the not needed foreground. For the given image the process of thinning needs 17 iterations to converge as seen in the figure 7.

The converged results from the Skeletonising and Thinning process as shown in the figure 4 and 7 does match the given images (b) and (c). Skeletonising is defined as the process of recursively removing the foreground region to reduce the given image to a skeletal remain which defines the extent and connectivity of the original region. This is nothing but the angular detail of the given image. So, when the given T alphabet image is Skeletonized till convergence, its dimension and connectivity are preserved in the output image in the form of two perpendicular straight lines defining T-image dimension and spurred line on the end points of straight lines defining the connectivity to original T-image corners. Thinning is the process of recursively removing the foreground region to reduce the given image to an eroded remain which defines the dimensions and only the extent of the original image. So, when the given T alphabet image is Thinned till convergence, its dimensions are preserved very faithfully in the output image in the form of two perpendicular straight lines.

The path 1[(a)->(b)->(c)] doesn't give the same result as the path 2[(a)->(c)], if the last step conversion process used was "Thinning" as shown in the figure 8. It so because of simple reason of the way thinning is defined and works, which is that it faithfully captures the dimension of the original image removing out the not needed foreground in the process. So, when thinning is applied to skeletonised image, it has no more dimension capture to do and leaves out the spurred lines as such. But if the process used was "Spur Removal" then the slanted lines in the skeletonised image are pruned off and thus we get the output resembling the 2[(a)->(c)]. This can be seen in the figure 9 of the results, yet there's still certain dissimilarity in the top centre region which is due to the different masking patterns for thinning and skeletonising process.

## DIGITAL HALFTONING

### Part(a): Digital Halftoning, Dithering Matrix

#### I. Abstract and Motivation

Digital image aren't reproducible in their original form of varying levels of intensity in the physical format because of the basic way the hardware like printers function. The printers function on a simple binary logic of 1's and 0's which intends put a black dot or not depending on the values given to print the images. So, it basically concurs to the point that the printers work only on binary image which are some sort replica of the original digital image. The process converting a grayscale digital image to a similar looking replica of binary image is known as Halftoning.

The printing of such a halftoned image is centred on the idea of representing a single graylevel by a group of dots which are very closely spaced, which ultimately gives an impression of viewing a grayscale image. So, it can be inferred that the resolution which equates to better quality of printed image depends on the density of binary image, which is always higher than original image. And it can also be inferred that the algorithm to generate such a binary image play a vital role in the halftoning process and ultimately affects the output image. The viability of each such algorithms can be easily tested by creating the binary output images of equal size to input gray scale image and the better algorithm should produce a more correlated output. The process is limited not only to grayscale image but extends to well over to colour images also, where in each of RGB components are treated as a separate entities

The two main ways to do halftoning is 1) the use of dithering matrix for adaptive thresholding and binary image generation, 2) error diffusion technique. The dithering matrix technique is based on the intuition that the varying levels of thresholds spread over a local image field, i.e. over pixel window creates increased gradient variation which makes the binary image more presentable and viewable with higher correlation to the original image. The amount of spread and the window size determines the effectiveness of the algorithm and consequently the quality of the output image. So, our motivation here is to study how such a pattern and size of dithering matrix affects the output image and derive best one among them.

The concept of halftoning can also be well extended into the areas of image compression and storage.

#### II. Approach and Procedures

1. *Apply given 4x4 matrix with the 4x4 Bayer Matrix and compare the results*

For the given I2, 2x2 Bayer matrix we have to generate I4, 4x4 Bayer matrix. For both given 4x4 matrix, A4 and 4x4 Bayer matrix, I4 we have to again generate threshold matrix, T4 and also normalise the given image for the process of comparison and consequent halftoning. The halftoning is done in following way

### a) 4x4 Bayer Matrix

**First**, generate 4x4 Bayer matrix, I4 from the given 2x2 Bayer matrix, I2 by the use of recursive matrix generation formula:

$$I_{2n}(x,y) = \begin{bmatrix} 4*I_n(x,y)+1 & 4*I_n(x,y)+2 \\ 4*I_n(x,y)+3 & 4*I_n(x,y) \end{bmatrix}$$

Given,

$$I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

Here the element values of the matrix indicate the likelihood of the pixel to be turned on, with 0 representing the higher probability and 3 the least. It can be easily seen that the elements which indicates likelihood are keyed in such that the lateral matrix neighbours have steep difference in probability and going by the recursive matrix formula generation, it can be deduced that the derivation higher order matrix also the follow the same suit of separation of near probabilities. These aides in creating binary image with

For all index value till 4

large gradient variation and thus, better output image.

For all index value till 4

$$I4[] = (4*I2[\text{index}\%2]) + I2[\text{index}/2]$$

Where,

I2 – the given 2x2 Bayer Matrix

I4 – the generated 4x4 Bayer Matrix

**Second**, from the generated index matrix create a threshold matrix T, which would be recursively used across the normalised image to help generate the binary image.

Formula:

$$T(x,y) = \frac{I(x,y) + 0.5}{N^2}$$

Where,

T(x,y) – Threshold matrix

I(x,y) – Bayer Matrix

N – Size of the Matrix

N<sup>2</sup> – Number of Pixels

$$T4[] = (I4[] + 0.5) / (4 * 4)$$

**Third,** normalise the given image which makes it occupy values between 0 and 1. This is done by dividing the grayscale of a pixel by maximum possible grayscale value of 255 and entire image is processed so on.

For all the pixels in the image  
 NormalisedImage[] =  
 OriginalImage[] / 255

**Forth,** Halftoned binary image is generated from the given normalised image and threshold matrix by window-wise thresholding and capping.

Formula:

$$G(i,j) = \begin{cases} 1 & \text{if } F(i,j) > T(i \bmod N, j \bmod N) \\ 0 & \text{else} \end{cases}$$

Where,

G(i,j) – The normalised grayscale value

F(i,j) – The original image grayscale value

For all

t  
h  
e  
p  
i  
x  
e  
l  
s  
i  
n  
t  
h  
e  
i  
m  
a  
g  
e  
i  
f  
(  
N  
o

r  
m  
a  
l  
i  
s  
e  
d  
I  
m  
a  
g  
e  
[  
i  
n  
d  
e  
x  
]  
>  
T  
[  
i  
n  
d  
e  
x  
]  
)  
t  
h  
e  
n  
B  
i  
n  
a  
r  
y  
I  
m  
a  
g  
e  
D  
a  
t  
a



[  
i  
n  
d  
e  
x  
]  
=  
1  
e  
l  
s  
e  
B  
i  
n  
a  
r  
y  
I  
m  
a  
g  
e  
D  
a  
t  
a  
[  
i  
n  
d  
e  
x  
]  
=  
0

This dithering matrix differs from the Bayer's matrix on the aspect of how its elements, which incidentally is the probability of pixel being turned on is distributed. It have the high probabilities placed in the centre with a less ones on the lateral neighbourhood and the least ones in the end points, thus more biasing at the centre which might create blocks of white and black creating a patchy binary image. And also due to lesser spread of near probabilities the binary image gradient would also be reduced, lessening the binary image quality. The matrix spread can be named as "spiral" spread.

IJSER

#### *b) 4x4 Given Spiral Matrix*

The given matrix is 4x4 already and so it voids the need of first step of matrix generation.

$$A_4(i,j) = \begin{bmatrix} 14 & 10 & 11 & 15 \\ 9 & 3 & 0 & 4 \\ 8 & 2 & 1 & 5 \\ 13 & 7 & 6 & 12 \end{bmatrix}$$

The steps are,

**First**, from the given A4 matrix create a threshold matrix T, which would be recursively used across the normalised image to help generate the binary image.

Formula:

$$T(x, y) = \frac{A(x, y) + 0.5}{N^2}$$

Where,

T(x,y) – Threshold matrix

A(x,y) – Given Spiral matrix

N – Size of the Matrix

N<sup>2</sup> – Number of Pixels

For all index value till 4

$$T4[] = (A4[] + 0.5) / (4 * 4)$$

**Second**, normalise the given image which makes it occupy values between 0 and 1. This is done by dividing the grayscale of a pixel by maximum possible grayscale value of 255 and entire image is processed so on.

For all the pixels in the image

$$\text{NormalisedImage[]} = \frac{\text{OriginalImage[]} }{255}$$

**Third**, Halftoned binary image is generated from the given normalised image and threshold matrix by window-wise thresholding and capping.

Formula:

$$G(i, j) = \begin{cases} 1 & \text{if } F(i, j) > T(i \bmod N, j \bmod N) \\ 0 & \text{else} \end{cases}$$

Where,

G(i,j) – The normalised grayscale value

F(i,j) – The original image grayscale value

For all

t  
h  
e

p  
i  
x  
e  
l  
s  
i  
n  
t  
h  
e  
i  
m  
a  
g  
e  
i  
f  
(  
N  
o  
r  
m  
a  
l  
i  
s  
e  
d  
I  
m  
a  
g  
e  
[  
i  
n  
d  
e  
x  
]  
>  
T  
[  
i  
n  
d  
e  
x  
]  
)

t	[
h	i
e	n
n	d
B	e
i	x
n	]
a	=
r	0
y	
I	
m	
a	
g	
e	
D	
a	
t	
a	
[	
i	
n	
d	
e	
x	
]	
=	
1	
e	
l	
s	
e	
B	
i	
n	
a	
r	
y	
I	
m	
a	
g	
e	
D	
a	
t	
a	

IJSER

Formula:

$$T(x, y) = \frac{I(x, y) + 0.5}{N^2}$$

2. Construct A8, an 8x8 matrix that has the same pattern with A4. Apply A8 and I8.

The halftoning process for each of the 8x8 matrix is similar to the ones applied to the 4x4 matrix. The 8x8 matrix has to be generated for each of the process and then the subsequent threshold matrix. The steps are,

a) 8x8 Bayer Matrix

**First**, generate 8x8 Bayer matrix, I8 from the given 2x2 Bayer matrix, I2 by the use of intermediately generating I4, 4x4 Bayer matrix by the use of recursive matrix generation formula:

$$I_{2n}(x, y) = \begin{bmatrix} 4 * I_n(x, y) + 1 & 4 * I_n(x, y) + 2 \\ 4 * I_n(x, y) + 3 & 4 * I_n(x, y) \end{bmatrix}$$

Given,

$$I_2(i, j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

We can notice that with the increased matrix size the spread of the matrix elements, i.e. the pixel probabilities are also spread more variedly and thus would enhance the output binary image gradient and consequently, the quality and the similarity to the original image.

Intermediate I4 Generation:

For all index value till 4

$$I4[] = (4 * I2[\text{index} \% 2]) + I2[\text{index} / 2]$$

I8 Generation:

For all index value till 8

$$I8[] = (4 * I4[\text{index} \% 4]) + I4[\text{index} / 4]$$

Where,

I2 – given 2x2 Bayer Matrix

I4 – intermediate 4x4 Bayer Matrix

I8 – generated 8x8 Bayer Matrix

**Second**, from the generated index matrix create a threshold matrix T, which would be recursively used across the normalised image to help generate the binary image.

Where,

$T(x,y)$  – Threshold matrix

$I(x,y)$  – Bayer Matrix

$N$  – Size of the Matrix

$N^2$  – Number of Pixels

For all index value till 8

$$T8[] = (I8[] + 0.5) / (8 * 8)$$

**Third**, normalise the given image which makes it occupy values between 0 and 1. This is done by dividing the grayscale of a pixel by maximum possible grayscale value of 255 and entire image is processed so on.

For all the pixels in the image

$$\text{NormalisedImage}[] =$$

$$\text{OriginalImage}[] / 255$$

**Forth**, Halftoned binary image is generated from the given normalised image and threshold matrix by window-wise thresholding and capping.

Formula:

$$G(i,j) = \begin{cases} 1 & \text{if } F(i,j) > T(i \bmod N, j \bmod N) \\ 0 & \text{else} \end{cases}$$

Where,

$G(i,j)$  – The normalised grayscale value

$F(i,j)$  – The original image grayscale value

For all

t  
h  
e  
p  
i  
x  
e  
l  
s  
i  
n  
t  
h  
e  
i

m  
a  
g  
e  
i  
f  
(  
N  
o  
r  
m  
a  
l  
i  
s  
e  
d  
I  
m  
a  
g  
e  
[  
i  
n  
d  
e  
x  
]  
>  
T  
[  
i  
n  
d  
e  
x  
]  
)  
t  
h  
e  
n  
B  
i  
n  
a  
r  
y  
I

m  
a  
g  
e  
D  
a  
t  
a  
[  
i  
n  
d  
e  
x  
]  
=  
1  
e  
l  
s  
e  
B  
i  
n  
a  
r  
y  
I  
m  
a  
g  
e  
D  
a  
t  
a  
[  
i  
n  
d  
e  
x  
]  
=  
0

### *b) 8x8 Spiral Matrix*

As, discussed earlier the given 4x4 matrix is generated in a manner such that the high probability elements placed in the centre with a little less ones on the lateral neighbourhood and the least ones in the end points. The arrangement is spiral in nature and the same technique can be used to intuitively generate A8 matrix after that the steps remain the same as in for 4x4 matrix.

Given 4x4 Spiral Matrix:

IJSER

For all index value till 8

$$T8[] = (A8[] + 0.5) / (8 * 8)$$

$$A_4(i,j) = \begin{bmatrix} 14 & 10 & 11 & 15 \\ 9 & 3 & 0 & 4 \\ 8 & 2 & 1 & 5 \\ 13 & 7 & 6 & 12 \end{bmatrix}$$

Inferred base 2x2 Spiral Matrix:

$$A_2(i,j) = \begin{vmatrix} 3 & 0 \\ 2 & 1 \end{vmatrix}$$

Extrapolated 8x8 Spiral Matrix

$$A_8(i,j) = \begin{vmatrix} 60 & 42 & 43 & 44 & 45 & 46 & 47 & 61 \\ 41 & 33 & 24 & 25 & 26 & 27 & 34 & 48 \\ 40 & 23 & 14 & 10 & 11 & 15 & 28 & 49 \\ 39 & 22 & 9 & 3 & 0 & 4 & 29 & 50 \\ 38 & 21 & 8 & 2 & 1 & 5 & 30 & 51 \\ 37 & 20 & 13 & 7 & 6 & 12 & 31 & 52 \\ 36 & 32 & 19 & 18 & 17 & 16 & 35 & 53 \\ 63 & 59 & 58 & 57 & 56 & 55 & 54 & 62 \end{vmatrix}$$

The steps are,

**First**, from the given A8 matrix create a threshold matrix T, which would be recursively used across the normalised image to help generate the binary image.

Formula:

$$T(x,y) = \frac{A(x,y) + 0.5}{N^2}$$

Where,

T(x,y) – Threshold matrix

A(x,y) – Given Spiral matrix

N – Size of the Matrix

N<sup>2</sup> – Number of Pixels

**Second**, normalise the given image which makes it occupy values between 0 and 1. This is done by dividing the grayscale of a pixel by maximum possible grayscale value of 255 and entire image is processed so on.

For all the pixels in the image  

$$\text{NormalisedImage}[] = \frac{\text{OriginalImage}[]}{255}$$

**Third**, Halftoned binary image is generated from the given normalised image and threshold matrix by window-wise thresholding and capping.

Formula:

$$G(i,j) = \begin{cases} 1 & \text{if } F(i,j) > T(i \bmod N, j \bmod N) \\ 0 & \text{else} \end{cases}$$

Where,

$G(i,j)$  – The normalised grayscale value

$F(i,j)$  – The original image grayscale value

For all

t  
h  
e  
p  
i  
x  
e  
l  
s  
i  
n  
t  
h  
e  
i  
m  
a  
g  
e  
i  
f  
(  
N  
o  
r

m  
a  
l  
i  
s  
e  
d  
I  
m  
a  
g  
e  
[  
i  
n  
d  
e  
x  
]  
>  
T  
[  
i  
n  
d  
e  
x  
]  
)  
t  
h  
e  
n  
B  
i  
n  
a  
r  
y  
I  
m  
a  
g  
e  
D  
a  
t  
a  
[



i  
n  
d  
e  
x  
]  
=  
1  
e  
l  
s  
e  
B  
i  
n  
a  
r  
y  
I  
m  
a  
g  
e  
D  
a  
t  
a  
[  
i  
n  
d  
e  
x  
]  
=  
0

& viewable. The segregation can be done logically in two ways:

- 1) Define three fixed threshold for the given image and let the thresholding be applied to the complete image, thus generating the 4 levelled pseudo binary image in a single pass
- 2) Image segregation based on scanning the grayscale value bits from MSB to LSB and separation of the grayscale value into levels by equitable allocation of the grayscale values to four different levels based on such a deterministic bit-wise scanning.

The first technique is applied here for the creation of the 4 levelled image, which requires the need for three different thresholds. The three thresholds need to be related to one another and derivable from other, to have some semblance of uniformity in the output. This can be done by using the same dithering matrix for the generation of all the three threshold matrices which are distinguishable from each other by a divisor factor, F. Since, the Bayer matrix

*3. Design a method to generate a printer-ready image of the Barbara image which is able to print 4 different intensity levels instead of 2.*

To design such a 4 level pseudo binary image, we need to segregate the grayscale values in the original image into 4 levels which comes down to generalising the principle of halftoning a bit further. From basic intuition one can infer that such a four levelled pseudo binary image would look more similar to the original image and also more appealing

seems to give better result for the halftoning than other given matrix we use Bayer matrix as a precursor. And, also inferring from the result that bigger the size of dithering matrix better the quality of the output image, we use the 8x8 Bayer matrix, I8 as a precursor.

The first step of I8 generation and second step of image normalisation is similar to the above techniques; the difference drops in only at the threshold matrix creation and thresholding stage. The steps are:

**First**, generate 8x8 Bayer matrix, I8 from the given 2x2 Bayer matrix, I2 by the use of intermediately generating I4, 4x4 Bayer matrix by the use of recursive matrix generation formula:

$$I_{2n}(x,y) = \begin{bmatrix} 4*I_n(x,y)+1 & 4*I_n(x,y)+2 \\ 4*I_n(x,y)+3 & 4*I_n(x,y) \end{bmatrix}$$

Given,

$$I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix}$$

We can notice that with the increased matrix size the spread of the matrix elements, i.e. the pixel probabilities are also spread more variedly and thus would enhance the output binary image gradient and consequently, the quality and the similarity to the original image.

Intermediate I4 Generation:

For all index value till 4

$$I4[] = (4*I2[\text{index}\%2]) + I2[\text{index}/2]$$

I8 Generation:

For all index value till 8

$$I8[] = (4*I4[\text{index}\%4]) + I4[\text{index}/4]$$

Where,

I2 – given 2x2 Bayer Matrix

I4 – intermediate 4x4 Bayer Matrix

I8 – generated 8x8 Bayer Matrix

**Second**, normalise the given image which makes it occupy values between 0 and 1. This is done by dividing the

grayscale of a pixel by maximum possible grayscale value of 255 and entire image is processed so on.

For all the pixels in the image

NormalisedImage[] =

OriginalImage[]/255

**Third**, from the generated index matrix create three threshold matrices T1, T2 and T3 which would be derivable from each other and differ from each other by the divisor factor, F. These would then be recursively used across the normalised image to help generate the binary image.

Formula:

$$T(x, y) = \frac{I(x, y) + 0.5}{N^2}$$

Where,

T(x,y) – Threshold matrix

I(x,y) – Bayer Matrix

N – Size of the Matrix

N<sup>2</sup> – Number of Pixels

F – Divisor Factor

For all index value till 8

T1\_8[] = (I8[]+0.5)/(8\*8)

T2\_8[] = T1\_8[]/F

T3\_8[] = T2\_8[]/F

**Fourth**, the four level pseudo binary image is then generated from the given normalised image and the three given threshold matrix by window-wise thresholding and capping.

For all the

pi

xe

ls

in

th

e

i

m

a

ge

if(

N

or

m

al

is

e

dI

m

a  
ge  
[i  
n  
d  
ex  
]  
>  
T  
1\_  
8[  
in  
d  
ex  
])

then 4LevelImageData[index]=3

if(NormalisedImage[index] < T1\_8[index]  
and NormalisedImage[index] >

T2\_8[index])

then 4LevelImageData[index]=2

if(NormalisedImage[index] < T2\_8[index]  
and NormalisedImage[index] >

T3\_8[index])

t

h  
e  
n  
4  
L  
e  
v  
e  
l  
I  
m  
a  
g  
e  
D  
a  
t  
a  
[  
i  
n  
d  
e  
x  
]

=	t
1	a
e	[
l	i
s	n
e	d
4	e
L	x
e	]
v	=
e	0
l	
I	
m	
a	
g	
e	
D	
a	

Thus the 4 level pseudo binary image can be generated. The divisor factor were changed to see the effect on the image. The divisor factor used: 2, 3, 1.5, 1.2, 1.

A better way to enhance it would be to vary the divisor factor, F depending on the image content such as the image with larger histogram spread can have larger divis

or factor and vice versa.

IJSER

#### IV. Discussion

The figure 1 shows the original grayscale Barbara image for which a printer ready binary image needs to be created. The process of halftoning using Dithering matrix technique has to be used here to generate such an image with the effects of varying size and pattern of the dithering matrix to be understood and later on based upon to create a better algorithm for such a halftoning process.

In the first part we use I4, 4x4 Bayer matrix and A4, 4x4 spiral matrix to generate the binary image by halftoning. In I4 Bayer matrix it can be easily seen that the elements which indicate probability of pixel being turned are keyed in a way that the lateral matrix neighbours have steep difference in probability and going by the recursive matrix formula generation, it can be deduced that the derivation higher

order matrix also follows the same suit of separation of probabilities of near value. This aids in creating binary image with large gradient variation and thus generating a better output image. This effect can be seen in figure 2 which shows the image halftoned by the I4 matrix, which is highly relatable to the original image and the image has a gradual change of texture without the presence of any blocky or patchy effect. Whereas when A4 given spiral matrix is used in which the high probabilities are placed in the centre with a lesser one on the lateral neighbourhood and the least ones in the end points. The biasing is more at the centre which creates blocks of white and black creating a patchy binary image. And also due to lesser spread of near probabilities the binary image gradient is also reduced, lessening the binary image quality. The change in the image texture as seen is steeper and the quality is thus drastically reduced.

The comparison can be seen here,



Figure 2: I4 Bayer Matrix

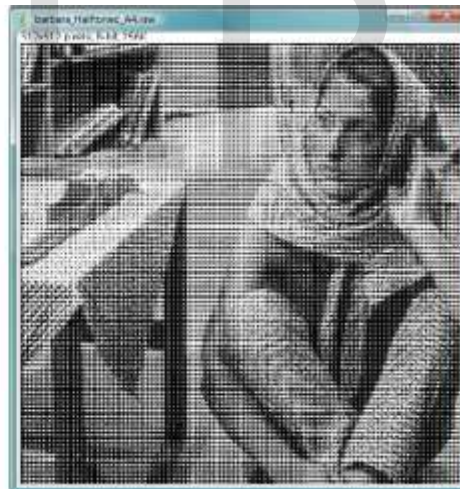


Figure 3: A4 Spiral Matrix

In the second part, we use increased size dithering matrix of 8x8 of both I8, Bayer and A8, given Spiral matrix. Here the effects of the smaller version are amplified. For I8 Bayer matrix, as the pixel probabilities are spread more variedly and in larger spatial dimensions it dramatically enhances the output binary image gradient and consequently, the quality and the similarity to the original image. Thus it can be concluded that greater the Dithering Matrix size better the output. But this axiom doesn't hold true for A8 Spiral matrix, in which the near probabilities are clumped more so together and the spread is limited. The image blockiness amplifies to greater extent and the image is rendered more unsuitable. The axiom now can be redefined as the that greater the Dithering Matrix size better the output for an algorithm which produces sufficiently good results even in the smaller scales and vice versa.

In third part of designing a method to generate 4-level pseudo binary image, we opted for case 1 where we used the I8 dithering matrix as the precursor for its stated benefit of better output image resolvability and viewability. Three thresholds were created based on the Divisor factor,  $F$  which can have value greater than 1 and on. With differential thresholding we generated images for each of the divisor factor of 2, 3, 1.5, 1.2 and 1. From the formula it can be inferred that greater the divisor factor  $F$ , greater is the range separation of thresholding. So, for images with broad histogram plot, we can obtain better output from larger divisor factor and vice versa. The given image had a narrow histogram plot more leaning towards the dark bounds, so consequently with large divisor factor of 2 and 3 we tended to get the output of inferior quality which can be seen in figures 7 and 8. The divisor factor of 1 was implemented just to test the efficacy of the algorithm, which by the way formula is structured ended up generating just binary images as seen in figure 10. The best result was obtained in the divisor factor of 1.2 as seen in figure 9, which has very large gradient variance and smooth varying texture. The algorithm thus can be improved by adaptively choosing the divisor factor depending in the image content and the histogram width.

The halftoning process by the use dithering matrix ends up losing lot of information due to hard and fast thresholding and throwing out the quantisation error, yet it defines an easy and effective way to generate binary images.

## Part(b): Digital Halftoning, Error Diffusion Techniques

### *a) Floyd-Steinberg's error diffusion with serpentine scanning*

#### I. Abstract and Motivation

Halftoning process by using the Dithering matrix leaves out lot of information from the original image pixel while thresholding, which ultimately results in creating the binary image which though has some semblance to original image and very much viewable imbibes lot of quantisation error and has more of artificial feel of texture to it.

Error Diffusion matrix for Floyd-Steinberg approach is given below, it can be noted that the matrices for the unidirectional and serpentine approach are symmetrical to each other as they tend to propagate error in opposite directions:

If by any means the quantisation error generated during the thresholding process can be propagated to the next part in weight-cascaded manner for the causal pixel of the pixel in consideration, much of the image information can be retained and the output image so generated would be of better quality and the output image would have a natural feel of texture. This process can be done by the means of error diffusion techniques, which aims to diffuse the error from the given pixel to the causal image pixels in a controlled and mathematically weighted manner.

The quality of the output binary image yet again depends on the algorithm, its parameters and the way it is implemented, which is due to the effect of how the error travels along the image. The parameters can be the size of the error diffusion matrix which decides how many causal pixels are involved in the diffusion process and the weight of the error diffusion matrix which decides how much of the error is distributed to the so called causal pixels. Our motivation here is to study all the above stated effects, understand their repercussions and design a better algorithm for halftoning using the error diffusion techniques.

#### II. Approach and Procedures

The basic steps in the Error Diffusion Techniques involve thresholding the image pixel in consideration with reference to a fixed threshold, calculating the quantisation error so generated and propagating it to neighbouring causal pixel in a mathematically weighted manner. This process is then recursively done for every other pixel in the image, with chaining mechanism being normal unidirectional or in a serpentine manner, which also defines to certain extent the output image quality due to the process of error accumulation and offloading.

Normal Unidirectional:

$$\frac{1}{16} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

Serpentine:

$$\frac{1}{16} \begin{bmatrix} 0 & 0 & 0 \\ 7 & 0 & 0 \\ 1 & 5 & 3 \end{bmatrix}$$

We are to apply only serpentine scanning in this method.

The Steps are,

**First**, the given image is boundary to 1 level each on all sides to properly apply the Error Diffusion Matrix. The image is then normalised which makes it occupy values between 0 and 1. This is done by dividing the grayscale of a pixel by maximum possible grayscale value of 255 and entire image is processed so on.

For all the pixels in the image  
 NormalisedImage[] =  
 OriginalImage[]/255

**Second**, create index for serpentine check and obtain the appropriate diffusion matrix for implementation. The column index can be created by iterating the condition for twice the width of the image and checking for column overrun, when it happens the column index is decremented from maximum till zero and in the same way when column overrun occurs the row index is incremented by one which is iterated only for half the height of the image to account for serpentine offset. Following the same suit, when the column overrun occurs the serpentine matrix can be chosen for implementation and normal one otherwise.

For half of the image height  
 For twice of the image width  
 If Columnindex>Imagewidth  
 Newcolumnindex=Image  
 width-Columnindex  
 Newrowindex=2\*rowind  
 ex+1  
 Implement Serpentine  
 matrix

Else

Newcolumnindex=  
 Columnindex  
 Newrowindex=2\*rowind  
 ex  
 Implement normal  
 matrix

**Third**, threshold the current image pixel under consideration with fixed threshold and calculate the quantisation error. Propagate this quantisation error in weighted manner based on the given error diffusion matrix to the neighbouring causal pixel.



Thresholding:

F

o  
r  
a  
l  
l  
t  
h  
e  
p  
i  
x  
e  
l  
s  
i  
n  
t  
h  
e  
i  
m  
a  
g  
e  
i  
f  
(  
N  
o  
r  
m  
a  
l  
i  
s  
s  
e  
d  
I  
m  
a  
g  
e  
[  
i  
n  
d  
e

x  
]  
>  
0  
.  
5  
)

Quanti  
sation  
Error=  
1-  
BinaryI  
mageD  
ata[ind  
ex]  
BinaryI  
mageD  
ata[ind  
ex]=1

else

Qua  
ntis  
atio  
n  
Erro  
r=  
Bina  
ryI  
mag  
eDat  
a[in  
dex]  
Bina  
ryI  
mag  
eDat  
a[in  
dex]  
=0

Error Propagation:

For all matrix size

NewImageData[Causal\_Pixels]=

ImageData[Causal\_Pixels] + (Quantisation

Error \* Error Diffusion Matrix[])

These steps are recursively applied for the complete image and hence the binary image is obtained.

*b) Jarvis, Judice and Ninke error diffusion*

Error Diffusion matrix for Jarvis, Judice and Ninke approach is given below, it can be noted that the matrices for the unidirectional and serpentine approach are symmetrical to each other as they tend to propagate error in opposite directions:

Normal Unidirectional:

$$\frac{1}{48} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

Serpentine:

$$\frac{1}{48} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 5 & 7 & 0 & 0 & 0 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

We are to apply both unidirectional normal and serpentine scanning in this method, which differs only in the second step of the process. The Steps are,

**First**, the given image is boundary to 2 levels each on all sides to properly apply the Error Diffusion Matrix which now is of 5x5 size. The image is then normalised which makes it

occupy values between 0 and 1. This is done by dividing the grayscale of a pixel by maximum possible grayscale value of 255 and entire image is processed so on.

```
For all the pixels in the image
    NormalisedImage[] =
        OriginalImage[]/255
```

**>>If the serpentine scan has to be applied then**

**Second,** create index for serpentine check and obtain the appropriate diffusion matrix for implementation. The column index can be created by iterating the condition for twice the width of the image and checking for column overrun, when it happens the column index is decremented from maximum till zero and in the same way when column overrun occurs the row index is incremented by one which is iterated only for half the height of the image to account for serpentine offset. Following the same suit, when the column overrun occurs the serpentine matrix can be chosen for implementation and normal one otherwise.

```
For half of the image height
    For twice of the image width
        If Columnindex>Imagewidth
            Newcolumnindex=Image
            width-Columnindex
            Newrowindex=2*rowind
            ex+1
            Implement Serpentine
            matrix
        Else
            Newcolumnindex=
            Columnindex
            Newrowindex=2*rowind
            ex
            Implement normal
            matrix
```

**Third,** threshold the current image pixel under consideration with fixed threshold and calculate the quantisation error. Propagate this quantisation error in weighted manner based on the given error diffusion matrix to the neighbouring causal pixel.

Thresholding:

```
F
o
r
```

a  
l  
l  
t  
h  
e  
p  
i  
x  
e  
l  
s  
i  
n  
t  
h  
e  
i  
m  
a  
g  
e  
i  
f  
(  
N  
o  
r  
m  
a  
l  
i  
s  
e  
d  
I  
m  
a  
g  
e  
[  
i  
n  
d  
e  
x  
]  
>  
0  
.  
5

```

)
    Quanti
    sation
    Error=
    1-
    BinaryI
    mageD
    ata[ind
    ex]
    BinaryI
    mageD
    ata[ind
    ex]=1
else
    Qua
    ntis
    atio
    n
    Erro
    r=
    Bina
    ryI
    mag
    eDat
    a[in
    dex]
    Bina
    ryI
    mag
    eDat
    a[in
    dex]
    =0

```

Error Propagation:

For all matrix size

NewImageData[Causal\_Pixels]=

ImageData[Causal\_Pixels] + (Quantisation

Error \* Error Diffusion Matrix[])

These steps are recursively applied for the complete image and hence the binary image is obtained.

### c) Stucki error diffusion

Error Diffusion matrix for Stucki approach is given below, it can be noted that the matrices for the unidirectional and serpentine approach are symmetrical to each other as they tend to propagate error in opposite directions:

Normal Unidirectional:

$$\frac{1}{42} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

Serpentine:

$$\frac{1}{42} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 4 & 8 & 0 & 0 & 0 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

We are to apply both unidirectional normal and serpentine scanning in this method, which differs only in the second step of the process. The Steps are,

**First**, the given image is boundary to 2 levels each on all sides to properly apply the Error Diffusion Matrix which now is of 5x5 size. The image is then normalised which makes it occupy values between 0 and 1. This is done by dividing the grayscale of a pixel by maximum possible grayscale value of 255 and entire image is processed so on.

For all the pixels in the image  
 NormalisedImage[] =  
 OriginalImage[]/255

**>>If the serpentine scan has to be applied then**

**Second**, create index for serpentine check and obtain the appropriate diffusion matrix for implementation. The column index can be created by iterating the condition for twice the width of the image and checking for column overrun, when it happens the column index is decremented from maximum till zero and in the same way when column overrun occurs the row index is incremented by one which is iterated only for half the height of the image to account for

serpentine offset. Following the same suit, when the column overrun occurs the serpentine matrix can be chosen for implementation and normal one otherwise.

For half of the image height

For twice of the image width

If Columnindex>Imagewidth

Newcolumnindex=Image  
width-Columnindex

```

        Newrowindex=2*rowindex+1
        Implement Serpentine
        matrix
    Else
        Newcolumnindex=
        Columnindex
        Newrowindex=2*rowindex
        ex
        Implement normal
        matrix

```

**Third,** threshold the current image pixel under consideration with fixed threshold and calculate the quantisation error. Propagate this quantisation error in weighted manner based on the given error diffusion matrix to the neighbouring causal pixel.

Thresholding:

F

o  
r  
a  
l  
l  
t  
h  
e  
p  
i  
x  
e  
l  
s  
i  
n  
t  
h  
e  
i  
m  
a  
g  
e  
i  
f  
(  
N  
o

r  
m  
a  
l  
i  
s  
e  
d  
I  
m  
a  
g  
e  
[  
i  
n  
d  
e  
x  
]  
>  
0  
.5  
)

Quanti  
sation  
Error=  
1-  
BinaryI  
mageD  
ata[ind  
ex]  
BinaryI  
mageD  
ata[ind  
ex]=1

else

Qua  
ntis  
atio  
n  
Erro  
r=  
Bina  
ryI  
mag  
eDat  
a[in

```

dex]
Bina
ryI
mag
eDat
a[in
dex]
=0

```

Error Propagation:

For all matrix size

NewImageData[Causal\_Pixels]=

ImageData[Causal\_Pixels] + (Quantisation

Error \* Error Diffusion Matrix[])

These steps are recursively applied for the complete image and hence the binary image is obtained.

*d) Suggest own approach to get better results*

From the results of the above techniques a number of points can be inferred. The output image quality improves when:

1) the Error Diffusion matrix so created which can diffuse maximum amount of error from the current pixel the nearest next pixel, ie the weights is more at the pixel point near the pixel under consideration. This can be seen by comparing the JJN and Stucki, wherein Stucki having such a more weight at very near causal pixel gives better and more dynamic output result.

2) the size of the error diffusion matrix is increased with the corresponding increase in the Error Diffusion matrix coefficients, this helps in diffusing the quantisation error to a broader region enhancing the dynamic range of output. This can be seen by comparing Floyd-Steinberg which is of 3x3 size and JJN & Stucki which are of 5x5 size, wherein JJN & Stucki give better output results as compared to the Floyd-Steinberg.

The approaches would be,

**First**, increase the weights for the 5x5 matrix as in:

$$1) \quad \frac{1}{56} \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 6 \\ 3 & 6 & 9 & 6 & 3 \\ 1 & 3 & 6 & 3 & 1 \end{vmatrix}$$

$$2) \quad \frac{1}{84} \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 16 & 8 \\ 4 & 8 & 16 & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \end{vmatrix}$$

This can then be applied in the normal unidirectional manner across the image to obtain the binary image. The implementation of such a matrix is shown in the figures 7 and 8.

**Second**, increase the size of the error diffusion matrix with appropriate increase in the weights in the matrix as in:

$$\frac{1}{104} \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 16 & 8 & 4 & 2 \\ 2 & 4 & 8 & 16 & 8 & 4 & 2 & 1 \\ 1 & 2 & 4 & 8 & 4 & 2 & 1 & 0 \\ 0 & 1 & 2 & 4 & 2 & 1 & 0 & 0 \end{vmatrix}$$

This again can be applied as the earlier matrices but the boundary extension now done till 3 levels. The implementation of such a matrix is shown in the figure 9.

There are other ways in which the error diffusion can be enhanced, they are:

**Third**, applying the halftoning process by means of error diffusion after the original image has been "contrast enhanced" by means of "histogram equalization". With the graylevel values of the original image spread for the entire range, i.e. with increased dynamic range the error generated

due to thresholding will be high and so more bias will be passed on to causal pixel, which will also be affected by large change therefore creating a binary image with high gradient variance, natural feel of texture and high correlation to the original image.

**Fourth**, this technique would be to choose the fixed threshold, which is normally 0.5 adaptively depending on the image content such as if the image is more bright the threshold



would have to be greater than 0.5 and lesser than 0.5 if the image turns out to be lighter. Such a dynamic choice of threshold though would bias the image but still give increased variance and dynamic range in the output binary image that would correspond to better improved quality.

Threshold  $\propto$  Mid grayscale of the image

**Fifth**, this approach is little too farfetched but going as per extrapolating the results we derived might give way better

### III. Experimental Results

Shown below are the results for the problem 3(b):



Figure 1: Original tea image



Figure 2: Floyd-Steinberg's error diffusion serpentine scanned tea image



Figure 3:



Figure 4

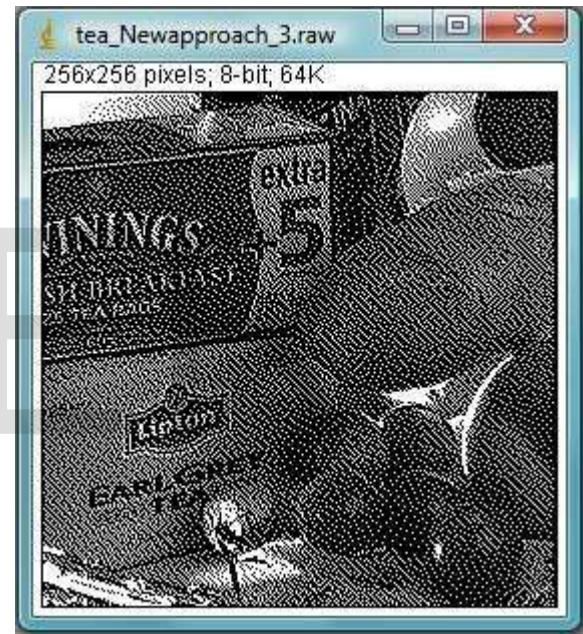
result. Here error diffusion is not pixel wise but block wise, each block representing the size of the Dithering matrix and obtain the result. Here s how the steps would go:

- 1) Calculate the quantisation error for the complete block pixel which are thresholded based on Dithering matrix like Bayer.
- 2) Use error diffusing matrix and super matrix and propagate the so obtained error from the pixels in a block by block basis
- 3) Repeat the process iteratively for the entire image



Figure 5 &amp;6

Figure 7: New Approach 3, enlarged error diffusion matrix scanned tea image



#### IV. Discussion

Figure 1 shows the original tea image which has to be halftoned using the error diffusion matrix. We are to find out the effect of change in the parameters like weights and window size and also the way the scanning process is done, on the output binary image.

result as given in the figure 2. Comparing the figure 2 and the figure obtained from the halftoning based on the dithering matrix. By Floyd-Steinberg's error diffusion approach we obtain he





The image obtained from the error diffusion technique is way better in terms of the original image semblance, viewability and gradient variance. The border extension by zeros helps absorb the excess of cascaded error variance in the end and removal of it won't make much a difference as the effect would only extend at the max to the nearest corner pixel which would be the border pixel. The serpentine scanning avoids the effect of the error being accumulated only to the right corner of the image and it also effectively helps diffuse the error equitably in all the directions. The capping of the so obtained pixel values to 1, when it overruns max is a moot point as the error propagated decays exponentially and therefore eliminating the need of such a capping mechanism.

Figures 7, 8 and 9 show the output binary image generated by the use of the new proposed approaches based on the tenets of the said two axioms that increased weight and increased size gives better results. Figure 8 which is formed from the matrix which has more weights as compared to the ones forming the figure 7, further proves the first axiom. But there's a limit to the increase in weight, if it extends above a set limit, the error propagation would

Figure 3 and 4, show the binary output images generated from by the use of JJN error diffusion technique in both normal unidirectional and serpentine scanning methods. Comparing this output image with the Floyd-Steinberg's gives a clear overview that the axiom of increased error diffusion window size with appropriate increase in weights gives a better output result. The difference between the serpentine and normal scanning though minimal is clearly obvious as seen from the images that the normal scanned as more divergence in the output binary image from the original image as it progresses to the right side of the image, which the serpentine scanning doesn't have. Figures 5 and 6, show the binary output images generated from the Stucki error diffusion matrix. The results of these are better than the ones obtained by JJN which has a diffusion matrix of similar size. This proves the second axiom that as the weights of the near causal pixel error propagation is increased the output becomes more dynamic and ressemblant.

change the causal pixel value more, creating extraneous noise. The figure 9 formed from the error diffusion matrix of larger size has better results from those of figures 8 and 7, proving second axiom of better result with increased size. Here again, there's limit to the increasing size which after a

## GEOMETRIC MODIFICATION

### Part(a): Geometric Modification, Puzzle Matching

#### I. Abstract and Motivation

Geometric modification is a very integral part of the Digital Image Processing technique and a core component of the Computer Graphics. It basically deals with manipulating with image in the aspects of its geometrical properties like size, rotation, position, projection and then on to other higher level aspects.

Bilinear interpolation is the corner stone of geometric image modification, for it allows the flexibility of tinkering with the pixel values at non-integer location, which always happens to be the case.

In the puzzle matching we learn how to wield the basic geometric modification technique and from two given dissonant pieces of varies geometric characteristic; create a wholesome, singular image.

#### II. Approach and Procedures

The given puzzle piece has following set of dissonance with the target tiger image and given are the corresponding action to be performed to overcome them and in step-wise manner patch the puzzle image to the target Tiger image with hole. The use of bilinear interpolation is implicit in each of the given steps involving geometrical modification.

##### 1) Rotation wrt to the reference axis

Process: Calculation of the corner points of the rotated image and from that derives the angle of rotation. With the calculated angle of rotation, rotate the puzzle piece wrt to the reference axis and negate the original rotation effect.

##### 2) Size mismatch between the puzzle piece and the hole in the target image

Process: Calculate the size of the hole in the target image; use

certain degree becomes useless overhead because causality doesn't extend till that limit. Use of contrast enhanced image would for sure increase the dynamic range and the output image quality. So, would be the adaptive thresholding but it's bias the image asymmetrically yet result would be excellent.

it to determine the scaling factor from the earlier found size of the puzzle piece. With the calculated scaling factor, downscale the image.

##### 3) Presence of extraneous white boundaries in the puzzle piece

Process: Calculate the exact boundary edges of the now down-scaled puzzle piece and prune the white outline by storing only the region of interest in a new image file.

##### 4) Affixing the puzzle piece in the exact hole position

Process: With the already pre-calculated hole coordinates, translational parameter is generated and the image is translated in reference to the Cartesian origin. The border of puzzle image is border extended to 2 pixel level to negate any pixel mismatch during the affixing process.

### *I. Border Finding and Width-Height-Angle calculation for the Puzzle piece*

1) Initialize count values for calculating the number of both white and black pixels in four directions for a set range.

countwup=countwdown=countwleft=countwright=0  
countbup=countbdown=countbleft=countbright=0

2) The background is white, so the for every non-white pixel found; the search is initiated for a range of 10 pixel points in all the four directions to find the number of white and black pixels.

For every hit  
Search top, bottom, right, left for 10 consecutive iterations  
If pixel found is white  
Increment the respective white counter  
If pixel found is black  
Increment the respective black counter

3) Now check the number of white and black pixel found in each such direction for each such hit. And depending on the following condition assign the marker pointer and store them as corner positions.

If top, right and bottom have a preset no. of white pixel and left has a preset no. of black Then it is top-right position

If top, left and bottom have a preset no. of white pixel and right has a preset no. of black Then it is bottom-left position

If top, right and left have a preset no. of white pixel and bottom has a preset no. of black Then it is top-left position

If left, right and bottom have a preset no. of white pixel and top has a preset no. of black Then it is bottom-right position

4) Now with the index of the corner positions find the height and width of the puzzle piece

PuzzleWidth=sqrt((downlefti-  
downrighti)^2+(downleftj-downrightj)^2)  
PuzzleHeight= sqrt((uplefti-

downlefti)^2+(upleftj-downleftj)^2)

5) Calculate the angle of the Puzzle piece tilt using the corner index values and deriving length and height of the puzzle piece project wrt the reference axis

y  
=  
d  
o  
w  
n  
r  
i  
g  
h  
t  
i  
-  
d  
o  
w  
n  
l  
e  
f  
t  
i

x  
=  
d  
o  
w  
n  
r  
i  
g  
h  
t  
j  
-  
d  
o  
w  
n  
l  
e  
f  
t  
j

position

## *II. Hole Finding and Coordinate location for the Puzzle piece*

It follows the same track as the puzzle border finding but the condition here is reversed as the hole is white pixel bound and rest of the image as different value variance, so a hit here would be finding a string of co-located white pixels. The condition for locating the corner point is also mildly changed due to the afore mentioned aspect and also due to the point that the hole lies in parallel with the reference axis.

1) Initialize count values for calculating the number of both white and black pixels in four directions for a set range.

countwup=countwdown=countwleft=countwright=0  
countbup=countbdown=countbleft=countbright=0

2) The background is a combinational value, so for every string of white pixel found; the search is initiated for a range of 10 pixel points in all the four directions to find the number of white and black pixels.

For every hit

Search top, bottom, right, left for 10 consecutive iterations

If pixel found is white

Increment the respective white counter

If pixel found is black

Increment the respective black counter

3) Now check the number of white and black pixel found in each such direction for each such hit. And depending on the following condition assign the marker pointer and store them as corner positions.

If top and right have a preset no. of white pixel & left and bottom has a preset no. of black Then it is bottom-left position

If top and left have a preset no. of white pixel & right and bottom has a preset no. of black Then it is bottom-right position

If bottom and left have a preset no. of white pixel & right and top has a preset no. of black Then it is top-left position

If bottom and right have a preset no. of white pixel & left and top has a preset no. of black Then it is top-right

4) Now with the index of the corner positions find the height and width of the hole in the target tiger image

HoleWidth= (downlefti-downrighti)

HoleHeight= (uplefti-downlefti)

### III. Puzzle piece rotation

1) Cartesian coordinate conversion and application of the rotational formula by using the angle generated above.

x  
=  
k  
-  
0  
.  
5

y  
=  
J  
+  
0  
.  
5  
-  
j

u  
=  
(  
x  
\*  
c  
o  
s  
(  
a  
n  
g  
l  
e  
)  
)  
+  
(  
y  
\*  
s  
i  
n  
(  
a  
n

g  
l  
e  
)  
)

v  
=  
(  
y  
\*  
c  
o  
s  
(  
a  
n  
g  
l  
e  
)  
)  
-  
(  
x  
\*  
s  
i  
n  
(  
a  
n  
g  
l  
e  
)  
)

q  
=  
u  
+  
0  
.  
5

p  
=  
P  
+  
0  
.  
5

-  
V

obtained value.

where,

P

-

I  
n  
p  
u  
t

i  
m  
a  
g  
e

h  
e  
i  
g  
h  
t

J

-

O  
u  
t  
p  
u  
t

i  
m  
a  
g  
e

h  
e  
i  
g  
h  
t

2) Apply bilinear interpolation for each of the RGB component of the input and output image and store the

#### IV. Puzzle piece scaling

1) Using the hole height or width and puzzle piece height or width, the scaling factor to down-scale the puzzle image is generated

$$\text{Scale} = \text{HoleHeight} / \text{PuzzleHeight}$$

2) The scaling factor is then applied to the image in the Cartesian perspective and using bilinear interpolation the output scaled image is generated.

x

=

k

-

0

.

5

y

=

J

+

0

.

5

-

j

$$u = (x / \text{scale})$$

$$v = (y / \text{scale})$$

q

=

u

+

0

.

5

p

=

P

+

0

.



### V. Boundary Pruning

Figure 1: Tiger Image with Hole

Using the boundary finding conditions again, the boundary of the now down-scaled puzzle piece is determined. Then just the puzzle piece is extracted and stored in an image file. This eases the process of translation and affixing on the target tiger image. The image file turns out to be of size 122X122.

### VI. Puzzle Translation and Affixing

1) The translational index for the puzzle image is just the top-left coordinates in terms of image coordinate system or bottom-left coordinates in terms of Cartesian coordinates; of the hole image found before. Since the pixel values are integer and translation is linear, it forgoes the use of bilinear interpolation. The image affixing is done in a wholesome manner for the iteration involving the hole in the target image only.

For all the index value within the hole  
`TigerImage[]=PuzzleImage[TranslatedIndex]`

Thus the puzzle problem is solved.

### III. Experimental Results

Following are the results for the problem 1(a):

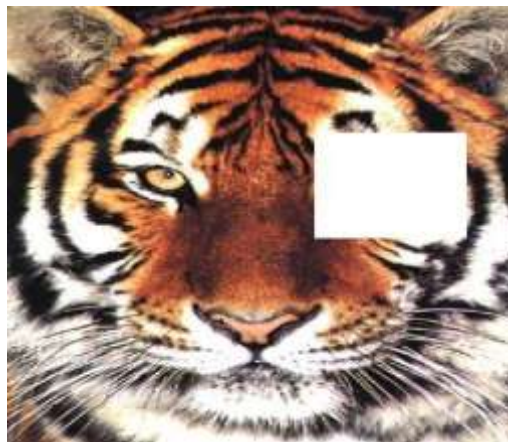




Figure 2: Puzzle Piece



Figure 3: Rotated Puzzle Piece



Figure 4: Down-Scaled Puzzle Piece (245X245)

of the downscaling before, would have further biased the image, cropping out the image quality.

The process could have been simplified by the use of affine transformation and so, consequently dolling out all the steps in the single execution, simplifying the entire procedure by just the matrix manipulation techniques.



Figure 5: Pruned Puzzle Piece (122X122)



Figure 6: Solved Tiger Image

#### IV. Discussion

The order in which the geometric transformation is carried out plays a vital role in the output image quality. If the puzzle image down-scaling was carried out at the first instance before the rotation, the output image after rotation for such an step would be more biased and affected by bilinear interpolation as the downscaling would have potentially reduced the image definition at each point and added more noise; and nonlinear rotation would have cascaded the effect.

Following the same suit, translation before the rotation and downscaling would have also proved to be a poor choice, as the translation to the high index value would require the rotation and scaling also to be carried out in that increased index value, increasing the time complexity. Also, the effect of introducing linear geometric transformation before the non-linear geometric transformation as seen in the example

As seen from the Figure 2 and 3, and Figure 3 and 4; geometric transformation adds basing to the output images due the inherent process of bilinear interpolation and the consequent input pixel weighting. The figure 6 shows the output, from which it can be seen that the fit is perfect and thus proving that geometric transformation is meticulously accurate. The boundary addition to the puzzle piece, enables the homogenous stitching

#### Part(b): Geometric Modification, Face Warping

### I. Abstract and Motivation

Image warping is the integral step for the many image processing techniques, like image morphing, that requires two or more sets of different images, aligned with respect to one other in one or more features. Image warping looks for the comparative features in the two given sets of images and tries to build an output set that the hybrid features of both the images. For example in image morphing, the set of features points like eye and ear has to be unambiguously aligned to ensure a very aesthetic output for the morphing technique.

The choice of feature, number of features and how are they being changed greatly affects the image warping outcome. Our motivation here is to study such a variation and learn how best to design the technique for best results.

### II. Approach and Procedures

The eyes of the two given input images of the man and tiger seems to almost well aligned and the ECP( eye centre point) also seems to well aligned. It'd be a moot point to choose them. The nose on the other hand for the two images is offset and the choice of it would not make the image more feature aligned but more symmetrical too.

The control point choice is nose, the coordinated for which is:

F  
o  
r

M  
a  
n  
:

j  
m  
=  
3  
0  
7  
  
k  
m  
=  
1  
9  
7

F  
o  
r  
  
T  
i  
g  
e  
r  
:

j  
t  
=  
3  
5  
7

k  
t  
=  
1  
9  
5

The output warping control point for this turns out to be:

$$jc=(jm+jt)/2=332$$

$$kc=(km+kt)/2=196$$

The said control point divides the images into four set of triangle each of which would be transformed by their own set of coefficients.

The two more sets of control point for each of the triangle are chosen to be the adjacent image corner points. The basic steps for the image warping is,

- 1) For each set of control point or a triangular region calculate the transform coefficients
- 2) For every given pixel during transformation determine in which triangular region it falls

3) Then transform the given pixel point using the coefficients specific to that triangle by the process of bilinear transformation.

### *I. Calculating the coefficients for the triangular region*

There are three set of coefficients are to be found;  $(a_0, b_0)$ ,  $(a_1, b_1)$  and  $(a_2, b_2)$ , from the three set of equation derived from the three set of control points for each input and output triangle.

The formula for this is given as

$$U = AX$$

Where,

$U$  –  $2 \times 3$  matrix; of the input control points  $(u_0, v_0)$ ,  $(u_1, v_1)$  and  $(u_2, v_2)$   $A$  –  $2 \times 3$  matrix; of the coefficients ;  $(a_0, b_0)$ ,  $(a_1, b_1)$  and  $(a_2, b_2)$

$X$  –  $3 \times 3$  matrix, of the outputs;  $(1, x_0, y_0)$ ,  $(1, x_1, y_1)$  and  $(1, x_2, y_2)$

The coefficient, hence can be calculated from the  $X$  matrix inversion and subsequent multiplication.

$$\text{Coefficient matrix, } A = X^{-1}U$$

### *II. Determination of the triangular region for each of the pixel location*

For image warping to work the pixel point from each triangular region must transformed accordingly wrt to their specific coefficients. The determination done by the “Barycentric Technique” which finds out if the given point is within the given set of triangle by finding the dot product of its vector with that of the other coordinates with one as reference and when the value turns out to be positive it is definitively said to lie within the said triangle

So, the given pixel point is iteratively search for triangular occupancy by comparing with all the given set of triangles. And the respective coordinates are chosen from then on and the transformation calculated.

Barycentric technique:

Here,

$A, B, C$  – are the triangular coordinates

$P$  – is the test point

V  
e  
c  
t  
o  
r  
  
C  
o  
m  
p  
u  
t  
a  
t  
i  
o  
n  
v  
0

=

C

-

A

v

1

=

B

-

A

v

2

=

D  
o  
t  
  
p  
r  
o  
d  
u  
c  
t  
  
c  
o  
m  
p  
u  
t  
a  
t  
i  
o  
n  
  
d  
o  
t  
0  
0  
  
=  
  
d  
o  
t  
(  
v  
0  
,  
  
v  
0  
)  
  
d  
o

t  
0  
1  
  
=  
  
d  
o  
t  
(  
v  
0  
,  
  
v  
1  
)  
  
d  
o  
t  
0  
2  
=  
d  
o  
t  
(  
v  
0  
,  
  
v  
2  
)  
  
d  
o  
t  
1  
1  
  
=  
  
d  
o  
t  
(

IJSER

v  
1  
,

determined, the respective coefficients are evoked and the transformation computed

$$U = AX$$

v  
1  
)

After which by the bilinear interpolation the output pixel values are computed and thus warped image is generated.

d  
o  
t  
1  
2

=

d  
o  
t  
(  
v  
1  
,  
v  
2  
)

IJSER

Barycentric coordinates computation

Denom = 1 /  
(dot00 \*  
dot11 - dot01  
\* dot01) u =  
(dot11 \*  
dot02 - dot01  
\* dot12) \*  
Denom v =  
(dot00 \*  
dot12 - dot01  
\* dot02) \*  
Denom

Checking if point is in the triangle

If (u >= 0) && (v >= 0) && (u + v < 1)  
Then P is in triangle ABC

### III. Pixel Transformation and Warping

When the triangular dependency of the given pixel is

### III. Experimental Results

Following are the results for the problem 1(b):

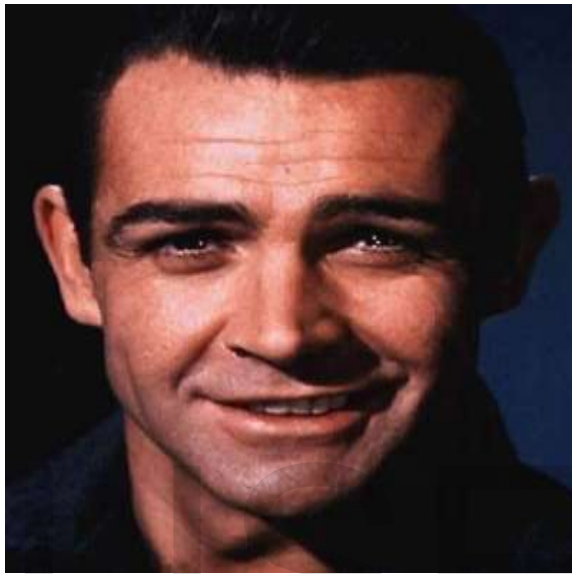


Figure 1: Original Man Image



Figure 2: Original Tiger Image



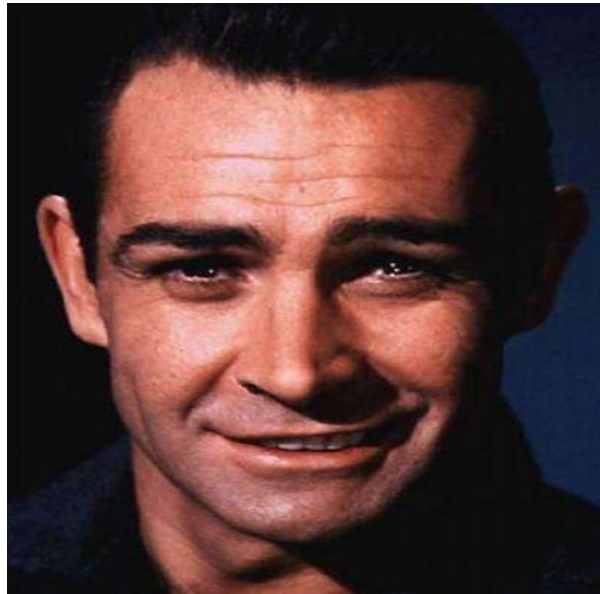


Figure 3: Warped Man Image



Figure 4: Warped Tiger Image

#### IV. Discussion

It can be clearly deduced that the choice of more control points would lend a better result, albeit would increase the computational efficiency. The choice of which feature is chosen to be the control point is also greatly deterministic in the quality of the output of the image. For example the choice of cheek or eyebrows is vague and is not that prominent of a characteristic. The best choice of features would be eyes, ECP, nose and ears.

The symmetry within the image also greatly affects the quality of warping as seen from the man warped image of figures 1 and 3 where the output is more symmetrical but the features doesn't quite align with warped tiger image in figure 4, due such an asymmetry within the man image. So, self symmetrical alignment of input image is bound to further improve the image warping quality. The choice for symmetrical alignment can be the median line of the ECP, nose and the chin; which highly localises and modulates the symmetrical aspect.

A comparison between the warped output images of the man and the tiger in figure 3 and 4 shows that now though the nose of both are well aligned, but the previously somewhat aligned eye feature is lost during warping. So care should be taken while accounting the choice of the control points.

## Part(c): Geometric Modification, Face Morphing

### I. Abstract and Motivation

Image morphing is a special effect technique. It is used very often in movie to show a set of transition between scenes or pull out contrasting dramatic effect. For example, transformation of a man into a supernatural being of some sort likes a werewolf. It is also used intensively for static images to create a sort of hybrid image from the images of two or more person. Apart from the entertainment, image morphing is also useful in the technology and scientific circle while studying some kind of transition, for example a epidemic spread mapping over a time scale.

Image morphing needs to be consistent and smooth for all the said techniques. Our motivation here is to study basic image morphing. Learn how it functions, analyse its flaws and try understanding how to build better Image Morphing mechanism.

### II. Approach and Procedures

The process of image morphing being done here is basic image cross-dissolving. It slowly transitions the amount of the one image content to the next, with incremental steps of the ingrained alpha value. The transition is done for each RGB channel and the output is combined to give morphed image.

The formula for the cross dissolving is given as:

$$I_{out}(i, j) = (1 - \alpha) \cdot I_{man}(i, j) + \alpha \cdot I_{tiger}(i, j)$$

$$0 \leq \alpha \leq 1$$

The process is;

- 2) Input both the Tiger and Man image
- 3) Apply cross-dissolving for each of the RGB channel separately for the output image.
- 4) Use three values of Alpha = 0.2, 0.5 & 0.8

Thus we obtain three man-tiger morphed image.

### III. Experimental Results

Following are the results for the problem 1(c):

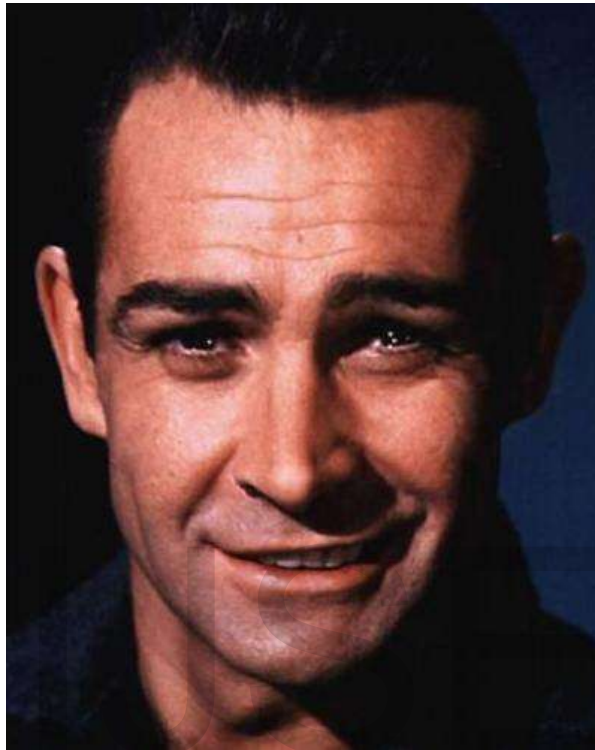


Figure 1: Warped Man Image

Figure 2: Warped Tiger Image





IJSER

Figure 3: Morphed Image, Alpha=0.2, Man to Tiger



Figure 4: Morphed Image, Alpha=0.5, Man to Tiger



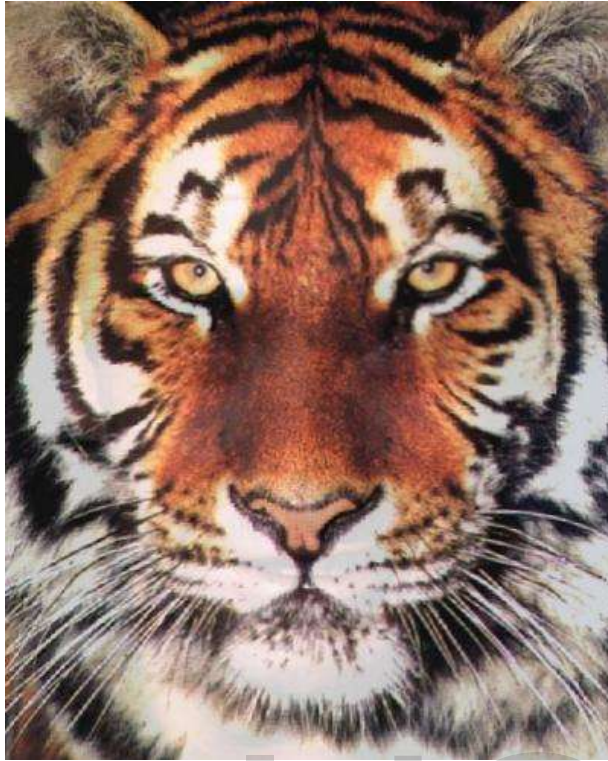


Figure 5: Morphed Image, Alpha=0.8, Man to Tiger

because of the cross-dissolving method applied, which basically tends to transform the RGB values from the man to tiger and so it happens that the dissonance occurs due to mismatch in the intermediaries due to the linear, non-pragmatic variability of the RGB values. This can be overcome by using the HSI colour space for the cross-dissolving. In which the Intensity can be linearly varied by the given set of formula but the Hue and Saturation has to be processed by using other set of formula for they being non-linear representation of RGB colour space and the given linear formula won't apply to it.

5)The next best method would be to generate single morphed output by the simultaneous warping-morphing, which might reduce the biasing effect and also have a proper cross-dissolution

#### IV. Discussion

From the figures 3,4,5 it is evident that the morphing results so obtained for each of alpha value of 0.2, 0.5, 0.8 is not smooth and has lot of inconsistencies. It can be because of

2 The choice of the alpha values is coarser and very largely separated. If the output images were obtained for close set of alpha values with small incremental, the morphing transition would have been smooth but it still would have suffered from the erratic output.

3 The first reason for such an unreasonable output is the imperfect alignment of the features for the warped man and the tiger images. This creates a series of convoluted intermediaries. This problem can be solved by having better warping mechanism, that is basically to have more control points and those control points should be reasonable feature points (like eyes or nose)

4 The second reason I believe that the output fails is

**Part(d): Geometric Modification, Back to 27****I. Abstract and Motivation**

Image morphing proves particularly interesting when the subjects are the time elapsed photo of the same person. The transition between the two allows us a peep into the effect of time on one's life.

Though the subjects apparently are the same, the image morphing still faces the same challenges of feature alignment, feature choice and proper cross dissolving. Our motivation here is to improve the morphing results better for the two such given subjects and analyse how better we can improve the results.

**II. Approach and Procedures**

For better morphing, the image warping must be better. And to improve the image warping we need to choose more control points and also ones that tend align the features maximally.

So, four extra control points are chosen: ECP (Eye Centre Point), Ends of mouth (smile) and Lower centre point to better align the features and also make the images more symmetrical. This breaks up the image into 9 sets of triangle as shown. The smile of them is in opposite direction and also the whole lower jaw is skewed, by taking the Mouth (or smile) end points as control points we can highly align the features. The lower edge symmetrises the image from within and the ECP is chosen to offset the drag effect from lower point warping as found in the Man-Tiger warping.

Triangular Break-down of the Image

The Control point coordinates were found to be:

Old Man:

ECP: (221,170)

Left Smile: (355,117)

Right Smile: (351,257)

Lower Point: (499,199)

Young Man:

ECP: (208,198)

Left Smile: (342,132)

Right Smile: (327,280)

Lower Point: (499,193)

is within the given set of triangle by finding the dot product of its vector with that of the other coordinates with one as reference and when the value turns out to be positive it is definitively said to lie within the said triangle

So, the given pixel point is iteratively search for triangular occupancy by comparing with all the given set of triangles. And the respective coordinates are chosen from then on and the transformation calculated.

Warped Image:

ECP: (215,183)

Left Smile: (349,125)

Right Smile: (339,269)

Lower Point: (499,196)

The Warping process is carried out as:

### *I. Calculating the coefficients for the triangular region*

There are three set of coefficients are to be found;  $(a_0, b_0)$ ,  $(a_1, b_1)$  and  $(a_2, b_2)$ , from the three set of equation derived from the three set of control points for each input and output triangle.

The formula for this is given as

$$U = AX$$

Where,

$U$  –  $2 \times 3$  matrix; of the input control points  $(u_0, v_0)$ ,  $(u_1, v_1)$  and  $(u_2, v_2)$

$A$  –  $2 \times 3$  matrix; of the coefficients ;  $(a_0, b_0)$ ,  $(a_1, b_1)$  and  $(a_2, b_2)$

$X$  –  $3 \times 3$  matrix, of the outputs;  $(1, x_0, y_0)$ ,  $(1, x_1, y_1)$  and  $(1, x_2, y_2)$

The coefficient, hence can be calculated from the  $X$  matrix inversion and subsequent multiplication.

$$\text{Coefficient matrix, } A = X^{-1}U$$

### *II. Determination of the triangular region for each of the pixel location*

For image warping to work the pixel point from each triangular region must transformed accordingly wrt to their specific coefficients. The determination done by the “Barycentric Technique” which finds out if the given point



v  
2

Barycentric technique:

=

Here,  
A, B, C – are the triangular coordinates  
P – is the test point

P

-

A

D  
o  
t

p  
r  
o  
d  
u  
c  
t  
c  
o  
m  
p  
u  
t  
a  
t  
i  
o  
n  
d  
o  
t  
0  
0  
=  
d  
o  
t  
(  
v  
0  
,

V  
e  
c  
t  
o  
r

C  
o  
m  
p  
u  
t  
a  
t  
i  
o  
n

v  
0

=

C

-

A

v  
1

=

B

-

A

IJSER

```

v
0
)

dot
0
1

=

dot
0
1
(
v
0
,

v
1
)

dot
0
2

=

dot
0
1
(
v
1
,

v
2
)

Barycentric coordinates computation
Denom = 1 /
(dot00 *
dot11 - dot01
* dot01) u =
(dot11 *
dot02 - dot01
* dot12) *
Denom v =
(dot00 *
dot12 - dot01
* dot02) *
Denom

Checking if point is in the triangle

```

If  $(u \geq 0) \&\& (v \geq 0) \&\& (u + v < 1)$   
 Then P is in triangle ABC

### *III. Pixel Transformation and Warping*

When the triangular dependency of the given pixel is determined, the respective coefficients are evoked and the transformation computed

$$U = AX$$

After which by the bilinear interpolation the output pixel values are computed and thus warped image is generated.

The Morphing by Cross-Dissolving is done as:

The process of image morphing being done here is basic image cross-dissolving. It slowly transitions the amount of the one image content to the next, with incremental steps of the ingrained alpha value. The transition is done for each RGB channel and the output is combined to give morphed image.

The formula for the cross dissolving is given as:

IJSER

$$I_{out}(i,j) = (1-\alpha) \cdot I_{mcm}(i,j) + \alpha \cdot I_{tiger}(i,j)$$

$$0 \leq \alpha \leq 1$$

The process is;

- 1) Input both the Old man and Young man image
- 2) Apply cross-dissolving for each of the RGB channel separately for the output image.
- 3) Use three values of Alpha = 0.2, 0.5 & 0.8

Thus we obtain three Old-Young man morphed image.

### III. Experimental Results

Following are the results for the problem 1(d):

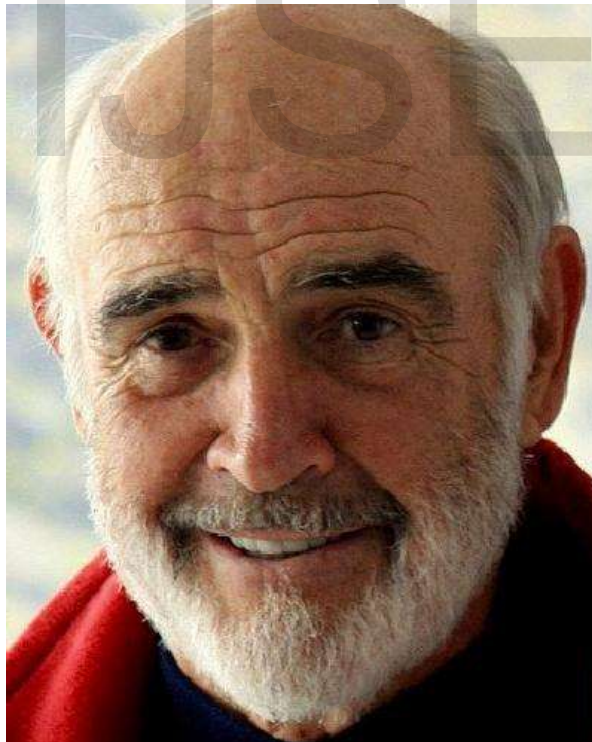


Figure 1: Old Man Image

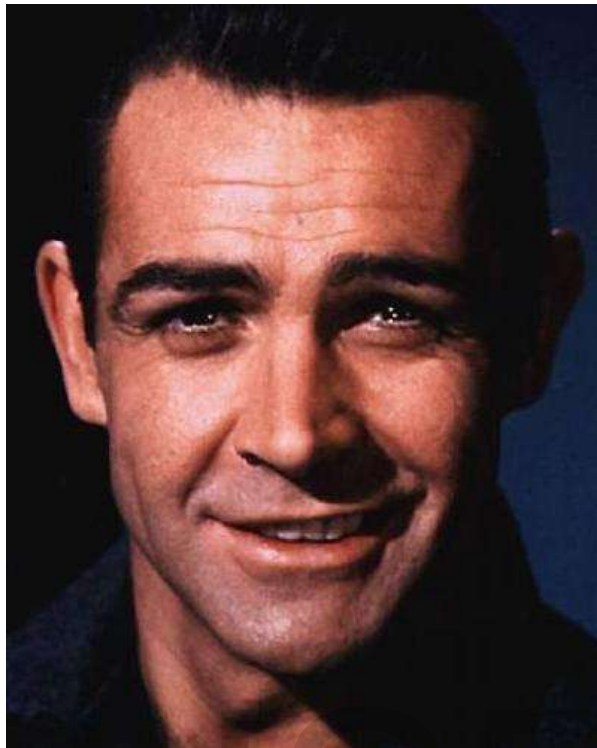


Figure 2: Young Man Image

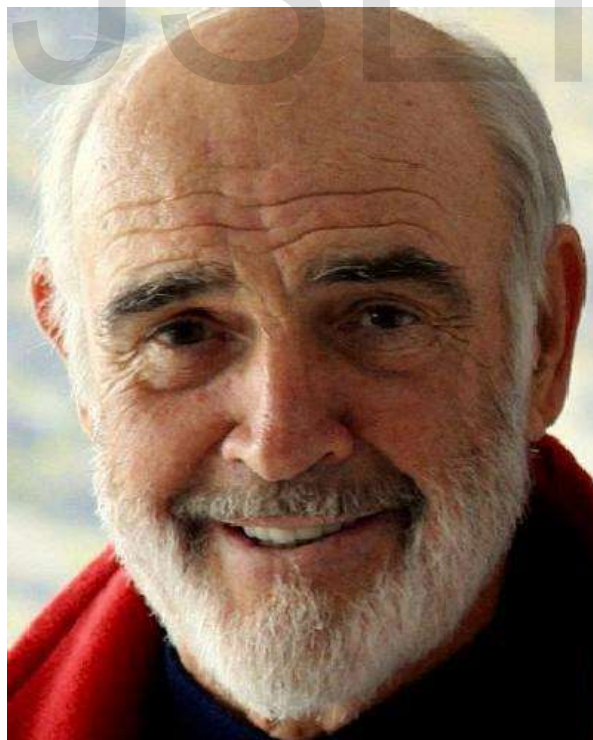


Figure 3: Warped Old Man Image



Figure 4: Warped Young Man Image

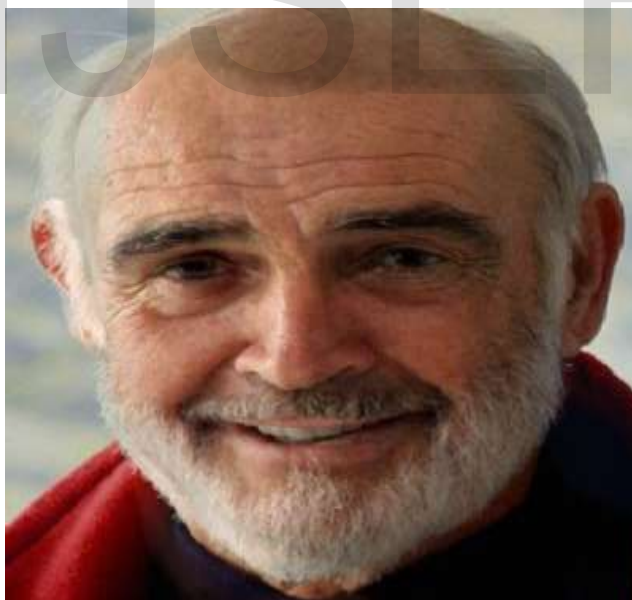


Figure 5: Morphed Image, Alpha=0.2, Old to Young

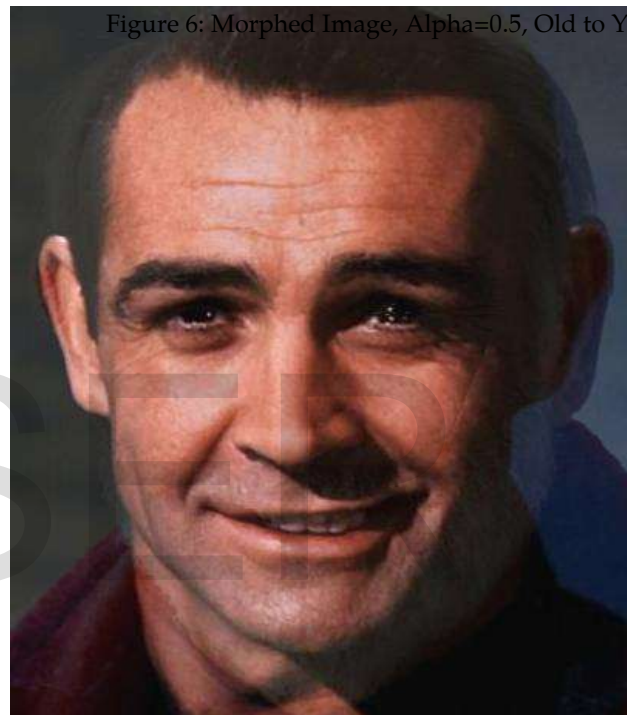


Figure 6: Morphed Image, Alpha=0.5, Old to Young

Figure 7: Morphed Image, Alpha=0.8, Old to Young

#### IV. Discussion

From the figures 5,6,7 it is apparent that the proper feature alignment does lead to better result for the image morphing, and hence we can put forth a general axiom that more the control points and better the feature alignment, better would be the morphing result. Albeit, the morphed image still suffers from certain inconsistency and has a large deficiency in terms of smoothness in transition. There are many causes for this:

- 1) The effect of differential lighting in both the images and contrast level. This can be partially reduced by letting the images by contrast enhanced and also, be greatly reduced if the intensity of the two images is equalized wrt each other at the initial point itself.
- 2) The reduced feature set alignment and thus the patchy effect.
- 3) The large step changes in alpha values, produces coarser results for comparison and to aesthetically view the transition

Yet, since the two subjects are the same person, we can see a clearer and better morphing result than compared to the one involving man-tiger transition.



## TEXTURE ANALYSIS AND SEGMENTATION USING LAW FILTER

**Part(a): Texture Analysis and Segmentation using Laws  
Filters, Texture Image Classification**

All of these set of kernels are inter tensored to give 9 5X5 Law filter to extract the image features.

The process is as follows:

### *I. Feature Extraction*

#### **I. Abstract and Motivation**

Texture is the feel or appearance of a surface, albeit it can't be described in absolute sense as to what constitute a texture and how well to define texture. The texture is a composite of many characteristics of the image like the intensity, location and variance of edges, spots, etc. and is unique in consistency pattern.

Texture classification is a milestone in image processing, though not a perfect art yet, because texture classification is one of the cornerstone of the image analysis and pattern recognition. By classifying the texture, we can club together images with similar texture and extrude out the rest, to help concentrate on the point of interest. For example, texture classification can be used in the remote sensing data to obtain data set specific to need.

The laws filtering technique proves to a great tool in texture analysis and can be used to classify large texture range. By extruding necessary feature values, we can classify the texture in different number of ways. Our motivation here is to implement and study the laws filtering for texture and learn the texture classification based on the feature values so derived.

#### **II. Approach and Procedures**

The laws filter is basically a set of kernels, either 3 or 5 length kernels which are used to derive a combination of texture feature extraction filters. The given law filter kernels for this situation is:

Level:

$$L5 = \frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Edge:

$$E5 = \frac{1}{6} \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \end{bmatrix}$$

Spot:

$$S5 = \frac{1}{2} \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \end{bmatrix}$$

1) Kernel Generation is done iteratively till combination of 9 Laws filters is generated by tensor product.

The iteration is kept as the conditional check for optimal filter combination and is given as:

Iteration 1:  $LL^T$

Iteration 2:  $LE^T$

Iteration 3:  $LS^T$

Iteration 4:  $EL^T$

Iteration 5:  $EE^T$

Iteration 6:  $ES^T$

Iteration 7:  $SL^T$

Iteration 8:  $SE^T$

Iteration 9:  $SS^T$

2) Feature vector generation for each pixel. The so generated Laws filter is applied iterative on each of the pixel value and thus creating a 9-dimensional feature vector for each pixel value

For image pixels

FeatureVector[9][index]=[LawsFilter[5X5]]\*{ImageValue[index]}

## II. Feature Averaging

The feature vector is then averaged for the entire image, thus creating a 9-dimensional vector associated with each of the 10 texture samples. The entire pixel feature value for each given feature is consolidated and averaged.

For all the samples

For all the features

For every pixel in the image

ConsolidatedFeature=FeatureVector[feature][index]

TextureFeatureVector=ConsolidatedFeature/Number of Image Pixel

## III. Classification

The Classification technique used here is the nearest neighbour classification, which groups the image with closest of the feature vectors. We have to generate four such groups from the 10 given samples.

1) Weighted Feature Vector Generation

A single consolidated feature vector is generated from the given 9-dimensional averaged feature vectors, by summing the product of the feature vectors and their weight. The weight is so assigned in the priority of that Spot>Edge>Level, as they amount of texture information so carried by them differs with level or intensity not giving out major information about texture.

Then sample belongs to given definite class  
Else

The weights are:

$LL^T : 1$   
 $LE^T : 2$   
 $LS^T : 2$   
 $EL^T : 2$   
 $EE^T : 3$   
 $ES^T : 4$   
 $SL^T : 1$   
 $SE^T : 4$   
 $SS^T : 5$

2) Calculation of pairwise distance of separation for each of the sample according to its weighted feature value. The Euclidean matrix is generated for this.

For all given sample pairs

$EuclideanMatrix[sample1][sample2] = |WeightedFeature[1] - WeightedFeature[2]|$

3) Generation of the Relation Matrix and Classification based on nearest neighbour from the Euclidean Matrix values. For each sample from the row values the nearest low value of sample is chosen and they clubbed together. This way a chain of relation matrix and classifying is done.

For all samples; 1

For all samples; 2

If  $EuclideanMatrix[1][2]$  lowest of all

$RelationMatrix[1]=2$

Then clubbing action is performed

First a definite class is formed where every element in relation matrix is related other element only in one-to-one basis

Definite class

For all samples

If givenSample=

$RelationMatrix[RelationMatrix[givensample]]$  Then Definite club

Second, the classes with one way linkage to definite clubs are added to the original club and the ones left out are classified in the last class

One way linkage

For all samples

If not definite club

If related to one sample of definite class

The sample belongs to leftover class

Thus the given 10 samples of texture images have been  
classified into 4 classes

### III. Experimental Results

Following are the results for the problem 2(a):

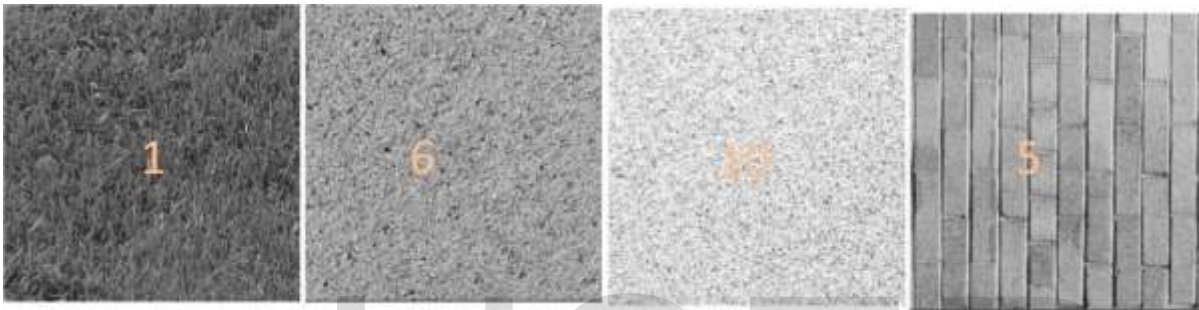


Figure 1: Texture Group 1 (1, 10, 5, 6)

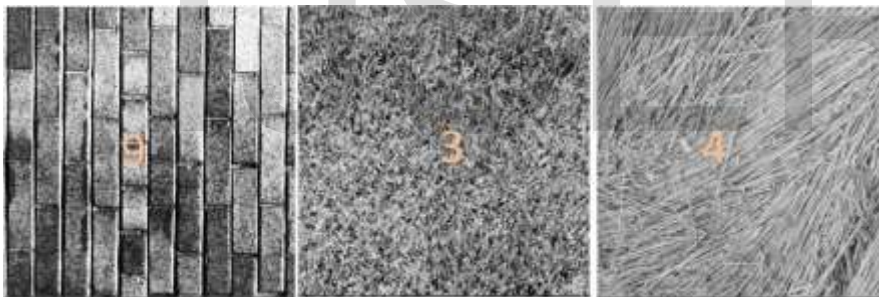


Figure 2: Texture Group 2 (3, 9, 4)

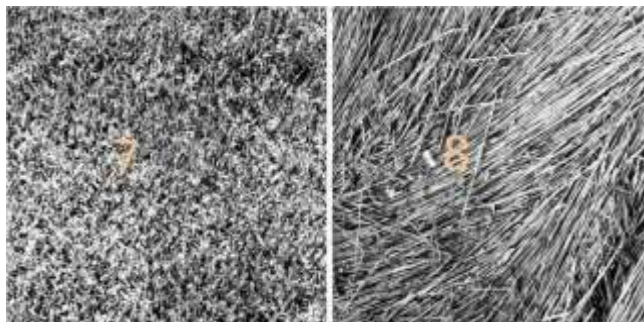


Figure 3: Texture Group 3 (7, 8)

Class 2: (5,9)

Class 3: (4,8)

Class 4: (2)



Figure 4: Texture Group 4 (2)

```

C:\Stuff\EE 569 Introduction to DIP\HW 3\Problem 2\Part 0\10_Clubbing Whole.exe
The Clubbed Images are:
1>>10>>5>>6>>
3>>9>>4>>
7>>8>>
2>>
-
  
```

Figure 5: Texture Classification Program Output

#### IV. Discussion

The given program classifies the 10 samples into 4 classes with certain amount of accuracy. The four classes so are as shown in figure 1, 2, 3, 4:

Class 1: (1,5,6,10)

Class 2: (3,4,9)

Class 3: (7,8)

Class 4: (2)

The actual classification from the perceptual point of view should have been

Class 1: (1,3,7,6,10)

images given.

As can be seen the Class 4 holds perfectly true and the Class 1 holds almost partially true but rest of the two classes is erroneous. The cause of this discrepancy and also certain level of accuracy can be explained as:

1) During the Weighted Feature vector calculation, most weightage was given to 2-D spot value, and a more weight to the combination of the spot and edge features. This was done to ensure that the unreliable level values was kept at minimum from playing any part and also, the majority of the texture information can be analysed from the way the edges were spread out and the spot were variant and distributed.

The Class 1 is inferred to have high density of miniscule spots and so the extra weight bounded them together in the feature space as the closest neighbour. But also the edge trace pattern for 1, 6, 10 looks most similar and extra weight to that feature has brought us partial correct value

2) The second highest weight was given to the two combination of the spot-edge, and has it turns out these filters tune to the local lateral or horizontal spot variation high amount of affinity defined by their filter build and the pattern for 9,3,4 follows such local maximal variation of spot and edged in orthogonal directionality. And the weighting again placed them as neighbours in the feature space, generating erratic value.

Though the edge pattern and level-edge combo pattern for 3,7 and 4,8 pair is highly correlative, the lower weightage to that feature space, placed them a bit further away in the feature vector space.

3) The lowest weightage given to the level and level combo, which is equitable to the intensity value; kept the wrong classification from creeping in. And that's the reason why, pattern 2 which though as similar intensity levels with other patterns, is kept a different group which is right class for it, as it doesn't relate to any given patterns here.

So, we can infer that though feature extraction works well, when coming down classification the algorithm doesn't prove to be highly effective has there's lack in knowledge as to how best define texture and which particular feature has more dominance in texture classification. So, the weighting and classification to a finer level can be done only by finetuning the algorithm depending on each set of input

## Part(b): Texture Analysis and Segmentation using Laws Filters, Texture Segmentation

Law filter to extract the image features. Then we use K-means algorithm based on energy computation to segment the image features.

The texture segmentation by K-means approach is as follows:

### I. Abstract and Motivation

Texture analysis is not only useful for separating out a set of images into different classes as previously seen, but its major advantage and power lies in the fact that it can differentiate and extract various texture components within an image. Such a tool is called texture segmentation, which is used to segment different area of image based in certain features. For example in a CAT scan image of brain, such a technique can used to isolate tumour or some benign abnormal growth. Not only that, it also forms an integral part of many image processing technique like OCR.

Our motivation here is to study how the feature vector response is when the Laws filter are applied to given image, the ways to classify the texture within the image, clustering and finally segment the images into different parts based on such an analysis. We also study the ways to further enhance and optimize this techniques, like using reduced feature space for better and efficient implementation.

### II. Approach and Procedures

The texture energy measures are computed by first applying small convolution kernels to a digital image, and then performing a nonlinear windowing operation. The convolution kernels are given as:

Level:

$$L5 = [1 \ 4 \ 6 \ 4 \ 1]$$

Edge:

$$E5 = [-1 \ -2 \ 0 \ 2 \ 1]$$

Spot:

$$S5 = [-1 \ 0 \ 2 \ 0 \ -1]$$

Wave:

$$W5 = [-1 \ 2 \ 0 \ -2 \ 1]$$

Ripple:

$$R5 = [1 \ -4 \ 6 \ -4 \ 1]$$

All of these set of kernels are inter tensored to give 25 5X5

### I. Feature Extraction

1) Kernel Generation is done iteratively till combination of 25 Laws filters is generated by tensor product.

The iteration is kept as the conditional check for optimal filter combination and is given as:

Iteration 1: $LL^T$	Iteration 2: $LE^T$	Iteration 3: $LS^T$	Iteration 4: $LW^T$
Iteration 5: $LR^T$	Iteration 6: $EL^T$	Iteration 7: $EE^T$	Iteration 8: $ES^T$
Iteration 9: $EW^T$	Iteration 10: $ER^T$	Iteration 11: $SL^T$	Iteration 12: $SE^T$
Iteration 13: $SS^T$	Iteration 14: $SW^T$	Iteration 15: $SR^T$	Iteration 16: $WL^T$
Iteration 17: $WE^T$	Iteration 18: $WS^T$	Iteration 19: $WW^T$	Iteration 20: $WR^T$
Iteration 21: $RL^T$	Iteration 22: $RE^T$	Iteration 23: $RS^T$	Iteration 24: $RW^T$
Iteration 25: $RR^T$			

2) Feature vector generation for each pixel. The so generated Laws filter is applied iterative on each of the pixel value and thus creating a 25-dimensional feature vector for each pixel value

For image pixels

FeatureVector[25][index]={LawsFilter[5X5]}\*{ImageValue[index]}

### II. Energy Computation

The energy computation is the mean of feature vector values about a pixel for a given window size for all the feature dimensions. The window size for the feature play a vital role in terms of segmentation, to track the effect the window size was varied from 7 till 17.

The formula for the energy computation is given as:

$$E(i,j) = \sum_m \sum_n | \text{FeatureData}(i+m,j+n) |^2$$

For  
o  
r

a  
l  
l

t

h  
e

f  
e  
a  
t  
u  
r  
e

d  
i  
m  
e  
n  
s  
i  
o  
n  
s

F  
o  
r  
a  
l  
l

t  
h  
e

i  
m  
a  
g  
e

P  
i  
x  
e  
l  
s

For the given window size

$E[Fdim][index]=$

$E[Fdim][index]+FeatureData[Fdim][windowindex]$

### III. Energy %ormalization



The level kernel  $L5L5^T$  is the only non-zero mean feature and doesn't give much information except for the contrast of the image, which hardly of any use in segmentation process. So, it is used to normalize the other feature vector values enabling easier manipulation and range efficiency.

For all the feature dimensions except  $L5L5^T$

IJSER

If PixelLabel=Current K value  
 ClusTotal=ClusTotal+E[index]  
 ClusterMean=ClusTotal/Total pixel in the Cluster

For all the image pixels  
 $E[Fdim][index] = E[Fdim][index] / E[L5L5^T][index]$

Recalculating the Euclidean Distance matrix for the mean value for each cluster:

#### IV. Segmentation

It consists of many sub-processes and the most significant of them is the K-mean calculation. The value of K defines the number of segments, which is 5 for the first image and 6 for the second image.

1) Centroid Initialization: The value of each of K centroid point during initialization plays a vital role for proper segmentation so the K-centroid values are initiated at the mid points of each of the supposed segment parts.

2) Euclidean Distance Calculation for every centroid point: The Euclidean distance for the consolidated feature vector values is calculated from the given centroid points and K-dimension Euclid Matrix is generated.

For Centroids  
 For all centroid-pixel pairs  
 $EuclideanMatrix[centroid][sample] = |E[Centroid] - E[Sample]|$

3) Initial Clustering and Labelling: From the Euclidean matrix so generated, the image pixels are segregated to K levels depending on which centroid they are closest to in the feature Euclid matrix and the pixel are according labelled from values to 1 and on for the later purpose of segmentation.

For all image pixels  
 If  $EuclideanMatrix[C][index]$  lowest of all  
 PixelLabel=CentroidIndex

4) Re-clustering: Here the image re-clustered by calculating new centroid, which iteratively tends to reduce the energy difference in a cluster

Mean Energy is calculated for each Cluster:

For all K values  
 For all image pixels

This produces reduced feature set of 20.

Set 2:

For all K values

For all pixel in the cluster

$\text{EuclideanMatrix}[\text{mean}][\text{index}] = |E[\text{mean}] - E[\text{index}]|$

New Centroid Calculation:

For all K values

For all pixel in the cluster

If  $\text{EuclideanMatrix}[\text{index}]$  is the lowest

$\text{Centroid} = \text{PixelLocation}$

5) The K-means algorithm is iterated till a convergence is reached, wherein the centroid doesn't move. For our problem it is usually around 60 iterations.

6) Image Segmentation and Storage: K different gray levels values are defines and for the given label of the pixel it segmented in the corresponding gray level

For all K values

For all pixel in the cluster belonging to current K

$\text{OutputImage}[\text{index}] = \text{GrayValue}[k]$

Thus the image is segmented into K regions of interest.

There are a set of features that are redundant or have symmetrical property to its orthogonal filters. The computational efficiency and space efficiency would be dramatically improved when we use reduced feature sets. The two type of reduced feature sets and the corresponding offsetting process is:

Set 1:

The Kernel involving L5 are highly symmetrical in their orthogonal pair and tend perform only the same function. So, it is possible to remove them use their orthogonal symmetric filters with added weight to be used for them.

The pairs are

$(LE^T, EL^T) (LS^T, SL^T) (LR^T, RL^T) (LW^T, WL^T)$

So one of the elements from each pair is removed and the other is weighed twice during energy consolidation.

The Kernel involving E5 are also symmetrical in their orthogonal pair and astonishingly perform the similar energy computation. So, it is possible to remove them use their orthogonal symmetric filters with added weight to be used for them.

The pairs are  
 $(E^T, SE^T)$   $(ER^T, RE^T)$   $(EW^T, WE^T)$

So one of the elements from each pair is removed and the other is weighed twice during energy consolidation.

This produces reduced feature set of 17.

### III. Experimental Results

Following are the results for the problem 2(b):

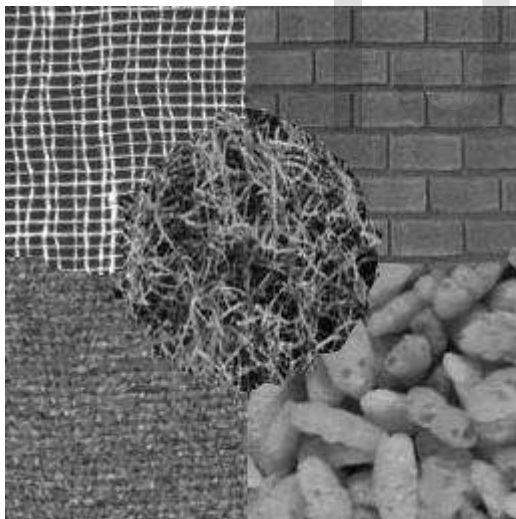


Figure 1: Given Image 1 with 5 Texture Features

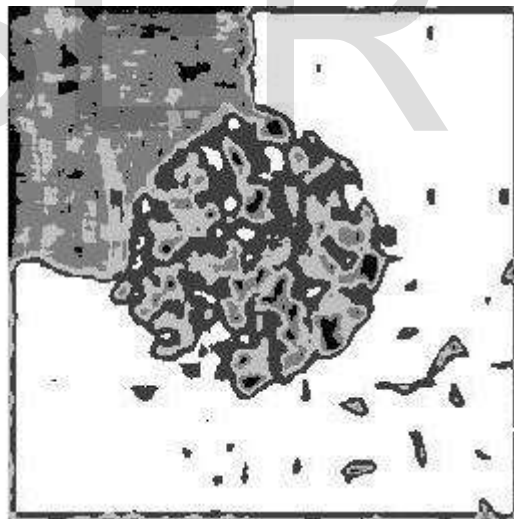


Figure 2: Segemeted Image 1; Window size 7



Figure 3: Segmented Image 1; Window size 9

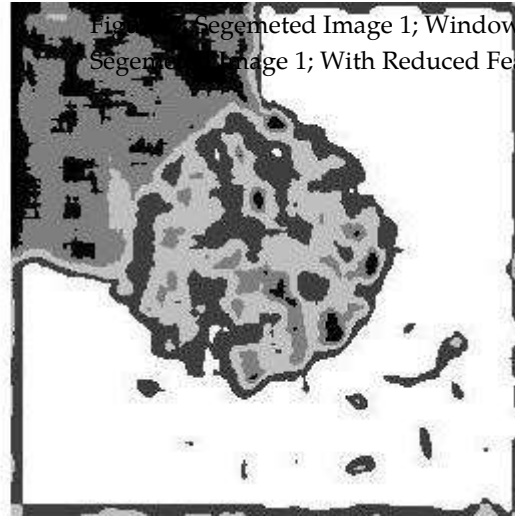


Figure 4: Segmented Image 1; Window size 11

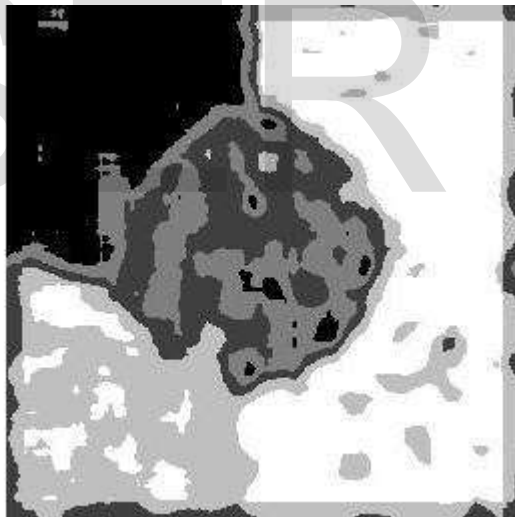
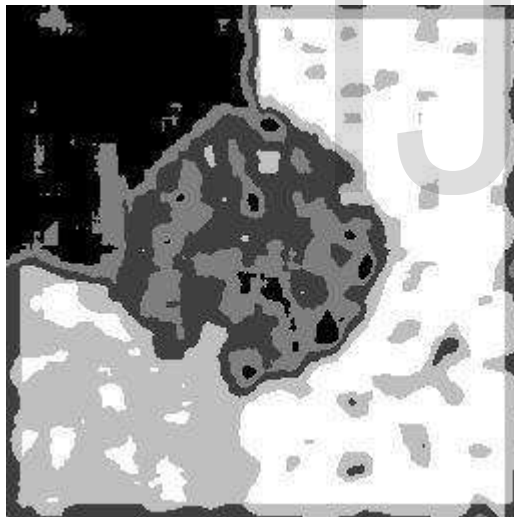


Figure 5: Segmented Image 1; Window size 13 Figure 6: Segmented Image 1; Window size 15 (Optimum)

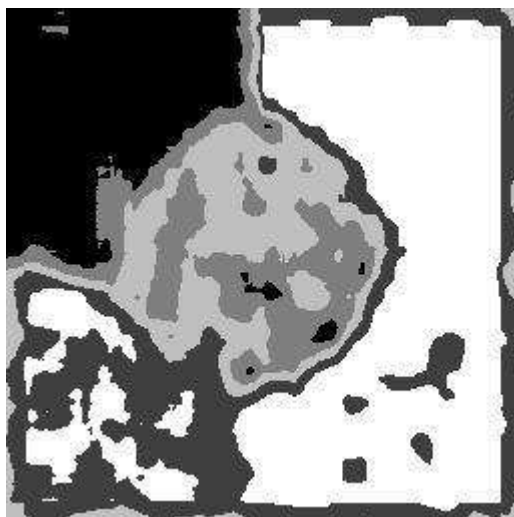




Figure 10: Given Image 2 with 6 Texture Features Figure 11: Segemeted Image 2; Window size 15 (Optimum)

Figure 9: Segemeted Image 1; With Reduced Features of 17

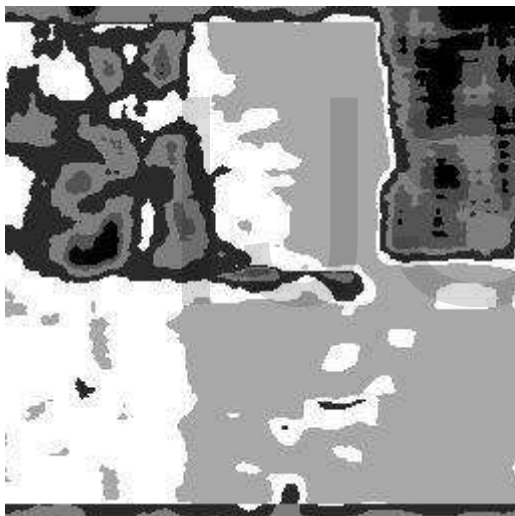
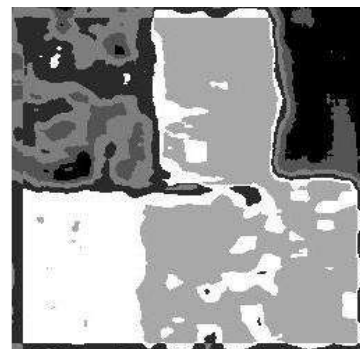


Figure 12: Segemeted Image 2; With Reduced

Figure 13: Segemeted Image 2; With Reduced Features of 20

Features of 17



more prominent and loss of segmentation visible. This is case in figure 9 for image 1 but not in figure 10 for image 2 because the texture content in image 1 were more in line with the edge variation and removal of them caused the loss of differentiability.

#### IV. Discussion

The window size for the energy computation plays a vital role in the texture based segmentation. The effects of this can be seen in the figures from 2 to 7 which has variance of window size from 7 to 17. If the window size is too small, the energy computation is highly localized and doesn't even out. This negatively affects the segmentation process has the energy of the particular cluster would take large time and sometime won't reach the lowest point and so two or more segments merge into one segment. This effect can easily seen in figure 2 of window size 7.

If the window size is too large the mean calculation for the energy extends well beyond the particular texture boundary and after a set number of iterations, the two texture would coalesce or centroid converge giving a false output, as seen in figure 7 of window size 17. The optimum window size is found to be 15 for the given scenario.

As seen in the figure 7 for K value 5 and figure 11 for K value 6, some segments seem to have partially merged and some segment have remnants of some other cluster. The initial value of centroid that's why plays an important part and the convergence result is highly dependent on their proper placement.

Also, certain feature vectors like wave and ripple have limited use in the given example whereas the spot, edge and combo as more relevance to the texture in the given image set. If more weight can be assigned to the later feature values, we might be able to improve our segmentation result dramatically. But yet again we can notice that the weight assignment and the filter choice is highly dependent on the image and needs to be tailored for each image. That is why it is difficult to segment out bricks and potatoes texture in both the given input images.

The reduced feature set of 20 for both images gives almost similar result to full feature as seen in 8 and 12 proving the point of orthogonal filter symmetry. But there's certain patchiness in the output along the vertical axis for both the images. This is explained by the fact that the removed features set were the horizontal Level combo and the image biased more on the vertical axis then. The same effect can be seen for the reduced feature size of 17, but the distortion is

## OPTICAL CHARACTER RECOGNITION

### Part(a): Optical Character Recognition, OCR Training

### Symbol Properties

The following set of symbol properties are calculated for each character:

- 1) Euler number, Area, Perimeter and Circularity

For this the bit quad technique is used, which extracts out the pixel number and their arrangement based on the quad pattern. The quad pattern is given as:

### I. Abstract and Motivation

Optical character recognition, OCR is the process of conversion of the images of the printed or handwritten text into machine encoded text. It used in a varied fields and vivid application, like a phone app to capture the image of the B-card and extract just the contact details. It is therefore useful in text mining, speech synthesis and other text conversion process and forms the intermediary step in most of the cases where a physical document is involved.

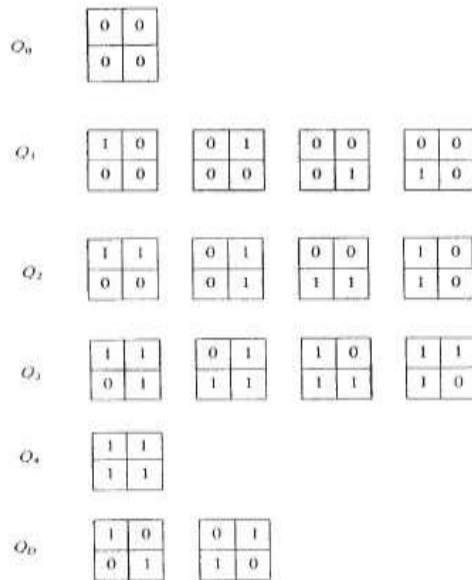
OCR is a highly complex technique and has to deal with lot of variability when the input is varying across a number of fonts, sizes, orientation, thickness and all the myriad properties. The simplest form OCR can be developed by training the algorithm to recognise particular set of unique character properties for the unique set of character given. The unique character can be its morphological, geometric, or any definable and unique properties. By such training, the OCR thus can recognize the given alphabets in any set of inputs. Our motivation here is to develop one such an efficient OCR algorithm by studying the properties of the characters given and building a decision tree around it, that would give the highest accuracy and effective recognition.

### II. Approach and Procedures

The approach to build a training set based OCR can be broken down to following set of procedures:

- 1) Understanding and calculating unique set of character properties. From this extracting the most unique and differentiable properties, and define every character based on that.
- 2) Build a decision tree, involving all the said unique properties in a manner that groups and segregates the characters logically and incorporates property having high intraclass variation at each step of the process.
- 3) Train the OCR for a number iterations on the given image set to achieve a certain level of accuracy and try to define and build decision tree, which at least in the starting point, is more generally applicable.





This is calculated for each pixel position and when a quad hit a found the respective counter is increased and stored.

For all the image pixel

If quad0 is hit Then increment quad 0 counter

If quad1 is hit Then increment quad 1 counter

If quad2 is hit Then increment quad 2 counter

If quad3 is hit Then increment quad 3 counter

If quad4 is hit Then increment quad 4 counter

If quadD is hit Then increment quad D counter

Then from the so found quad pattern hits the properties are as calculated:

Area:

$$A_O = \frac{1}{4} [n\{Q_1\} + 2n\{Q_2\} + 3n\{Q_3\} + 4n\{Q_4\} + 2n\{Q_D\}]$$

Perimeter:

$$P_O = n\{Q_1\} + n\{Q_2\} + n\{Q_3\} + 2n\{Q_D\}$$

Euler No.: (8-connectivity)

$$E = \frac{1}{4} [n\{Q_1\} - n\{Q_3\} - 2n\{Q_D\}]$$

Circularity:

$$C_O = \frac{4\pi A_O}{(P_O)^2}$$

## 2) Aspect Ratio and normalised Area and Perimeter

From the bound index values, the midpoint index of the character is found and from there on the horizontal and vertical index for it is found.

$$\text{RowMid} = (\text{RightBound} + \text{LeftBound}) / 2$$

$$\text{ColumnMid} = (\text{BottomBound} + \text{TopBound}) / 2$$

From this for all the image pixel equal points about RowMid and ColumnMid is found and normalised by the area covered and the percentage of it gives the Vertical and Horizontal symmetry respectively for Column and Row mirror search

For this the bounding box for each character is to be recognised and that can be done by running four searches for each of the combination of the image search.

Top-Bottom and Left- Right  
Image search, for all image  
pixels If PixelValue=1  
LeftBound is the lowest possible index value for column  
index

Top-Bottom and Right-Left  
Image search, for all image  
pixels If PixelValue=1  
RightBound is the highest possible index value for column  
index

Left- Right and Top-Bottom  
Image search, for all image  
pixels If PixelValue=1  
TopBound is the lowest possible index value for row index

Top Left- Right and Bottom-Top  
Image search, for all image pixels If  
PixelValue=1  
BottomBound is the highest possible index value for row  
index

From these values the Bounding box area and perimeter is calculated:

$$\text{Width} = (\text{RightBound} - \text{LeftBound})$$

$$\text{Height} = (\text{BottomBound} - \text{TopBound})$$

$$\text{BoundingArea} = \text{Width} * \text{Height}$$

$$\text{BoundingPerimeter} = 2 * [\text{Width} + \text{Height}]$$

From this,

$$\text{NormalisedArea} = \text{Area} / \text{BoundingArea}$$

$$\text{NormalisedPerimeter} = \text{Perimeter} / \text{BoundingPerimeter}$$

$$\text{AspectRatio} = \text{Width} / \text{Height}$$

### 3) Symmetry

## 3) Row and Column shift based on Centre of Gravity

## 4) Spatial Moment

It can be used to find the first order column and row moment, zero and second order moment from then on can be used to calculate the centroid points. Which is as follows

But to further solve the problem with OCR generality (10 Characters) and create an accurate OCR we need one more property:

Zero Order:

$$M(0, 0) = \sum_{j=1}^J \sum_{k=1}^K F(j, k)$$

First order row moment:

$$M(1, 0) = \frac{1}{J} \sum_{j=1}^J \sum_{k=1}^K x_j F(j, k)$$

First order column moment:

$$M(0, 1) = \frac{1}{K} \sum_{j=1}^J \sum_{k=1}^K y_k F(j, k)$$

The centroid points (centre of gravity) from it are calculated as:

$$\bar{x}_j = \frac{M(1, 0)}{M(0, 0)}$$

$$\bar{y}_k = \frac{M(0, 1)}{M(0, 0)}$$

This can be used to measure the shift between the Row and Column mid values and Centroid point essentially giving us the opportunity to measure the spread of the given character.

The shift is calculated as:

RowShift=  $x_j$  – RowMid

ColumnShift=  $y_k$  – ColumnMid

From all the calculated values only three values turn out to be highly unique and more general they are:

- 1) Euler Number
- 2) Symmetry

Using this we divide the character set into three classes:

*So, I define three new properties: Verticality, Horizontality and VHratio*

Class 1, Euler no = -1: 8, since there's only one character with such an affinity we can print this one with 100% accuracy

#### 6) Verticality, Horizontality and VH ratio

Verticality is defined as the number of pixel strings (of two consecutive) which lie parallel to vertical axis. This would be high for character like: M, N and low for Z, S

Horizontality is defined as the number of pixel strings (of two consecutive) which lie parallel to horizontal axis. This would be high for character like: Z, 2 and low for M, N

VHratio is the ratio of Verticality to the Horizontality.

The verticality and horizontality can be found by using the "Bit Quad" patterns themselves, which are:

For vertical;

$Q_2$

0	1	1	0
0	1	1	0

IJSER

For horizontal,

$Q_2$

1	1	0	0
0	0	1	1

For each hit the respective count is increased and the total is then normalised wrt to the Area.

VH ratio gives more scale invariant value to the character definition and so, it can be used for OCR decision tree very smoothly.

### Decision Tree

#### 1. Euler number

First the Euler number for the given character is found out and for the training set it can be -1, 0, 1

Class 2, Euler no=0: P, R, 9

Class 3, Euler no=1: 5, M, N, S, 2, Z

The last two classes has to further undergo analysis for proper differentiation.

## II. Class 2

The class 2 is then differentiated based on a) Spatial Moment (Shift) and b) Symmetry

It has follows:

### a) Spatial Moment (Shift)

The character P has more mass (hence more shift) towards top, R towards left and 9 towards right. By careful analysis the general values for these are determined and the weight for each is assigned:

If RowShift<-8  
 $P \gg W_p = W_p + 2$

If Columnshift>0  
 $9 \gg W_9 = W_9 + 2$

If RowShift<-1 && Columnshift<-1  
 $R \gg W_r = W_r + 2$

The weights are assigned for finding how accurately the character is determined and also places **probability** in our own decision process, making the process more logical.

### b) Symmetry

The Character 9 is relatively more symmetrical in both axis and the Character P the least. So the Assignment is:

If versym<50 && horsymp<70  
 $P \gg W_p = W_p + 1$

I  
f

v

e  
r  
s  
y  
m  
>  
7  
5  
&  
&  
h  
o  
r  
s  
y  
m  
p  
>  
8  
5  
  
9  
>  
>  
W  
9  
=  
W  
9  
+  
1  
  
e  
l  
s  
e  
  
f  
o  
r  
  
m  
i  
d  
  
v  
a  
l  
u  
e  
s  
  
R  
>

>  
W  
r  
=  
W  
r  
+  
1

The classification is then done by analysing the weight and finding out the percentage of accuracy from the weight also:

IJSER

If Versym<63

5>>W5=W5+1

If Wp is large Character P

If Wr is large Character R

If W9 is large Character 9

If versym>70&&horsym>75

S>>Ws=Ws+1.5

Else for all other values

The percentage is calculated as the ration of the weight of the found character to the total weights of all the character, this gives a measure of the OCR belief.

### III. Class 3

The class 3 since has more character is then differentiated based on a) Spatial Moment (Shift) b) Symmetry and VH ration(which is highly reliable for this class)

It has follows:

#### a) Spatial Moment (Shift)

The character 5 has more mass (hence more shift) towards bottom and 2 towards bottom and N towards left. By careful analysis the general values for these are determined and the weight for each is assigned:

If Columnshift<0

n>>Wn=Wn+0.5

if Rowshift<0&&Columnshift>0

Z>>Wz=Wz+0.5

2>>W2=W2+0.5

The weights are assigned for finding how accurately the character is determined and also places **probability** in our own decision process, making the process more logical.

#### b) Symmetry

The Character M is highly symmetrical in vertical axis, Character 5 the least along vertical and rest have nominal value. So the Assignment is:

If horsym>85

M>>Wm=Wm+2

$N \gg W_n = W_n + 0.33$

$2 \gg W_2 = W_2 + 0.33$

$Z \gg W_z = W_z + 0.33$

c) VH Ratio

The Verticality is high for N, M and Horizontality for 2, Z. So the VH ration for N, M is very large and very small for 2, Z. The character S and 5 posses very little relativity. So the assignment is:

if VHratio>2

$N \gg W_n = W_n + 1.5$

$M \gg W_m = W_m + 1$

If VHratio<0.58

$Z \gg W_z = W_z + 1.5$

E  
l  
s  
e

f  
o  
r

o  
t  
h  
e  
r

v  
a  
l  
u  
e  
s

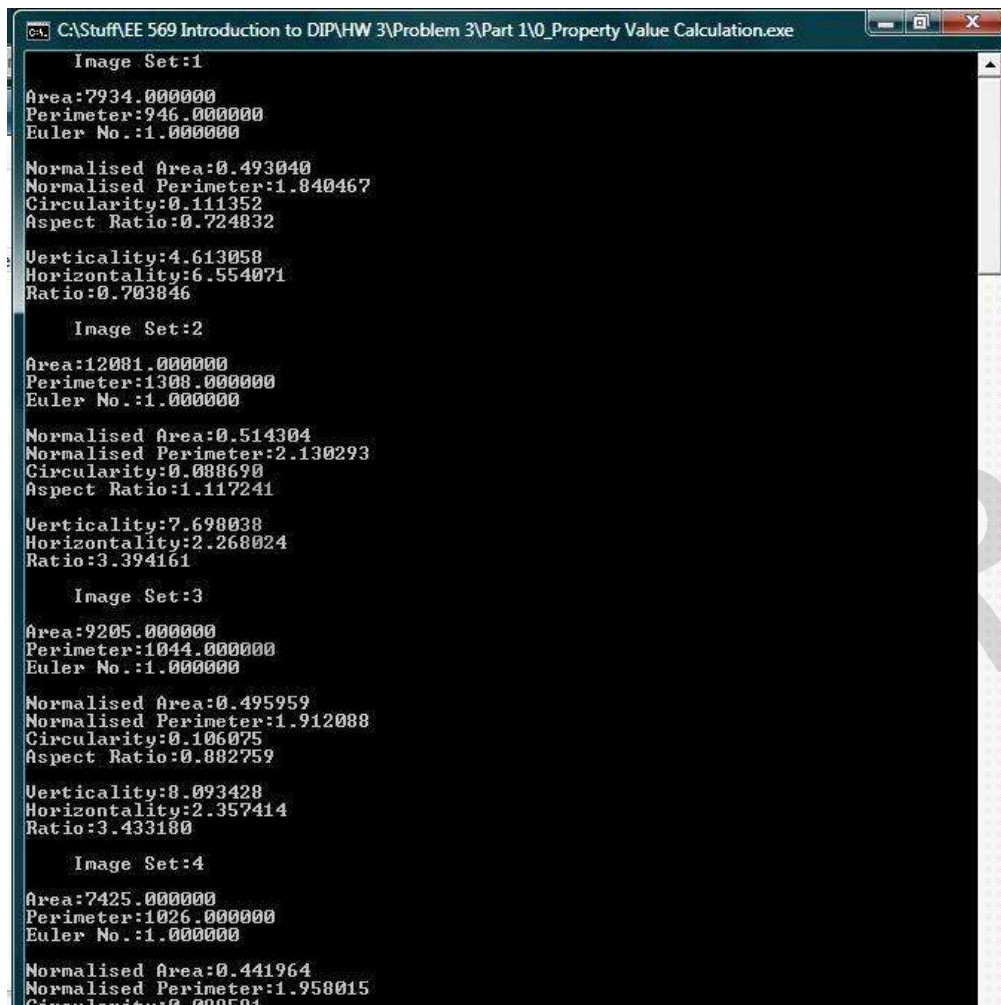
2  
>  
>  
W  
2  
=  
W

2  
+  
0  
.  
6  
6  
  
5  
>  
>  
W  
5  
=  
W  
5  
+  
0  
.  
3  
3  
  
S  
>  
>  
W  
s  
=  
W  
s  
+  
0  
.  
3  
3  
  
The classification is then done by analysing the weight and finding out the percentage of accuracy from the weight also:  
  
If W5 is large Character 5  
If Wm is large Character M  
If Wn is large Character N  
If Ws is large Character S  
If W2 is large Character 2  
If Wz is large Character Z  
The percentage is calculated as the ration of the weight of the found character to the total weights of all the character, this gives a measure of the OCR belief.  
Thus the OCR is developed and trained on the Training image set and also can be made generally applicable for the other imager set.



### III. Experimental Results

Following are the results for the problem 3(a):



```

C:\Stuff\EE 569 Introduction to DIP\HW 3\Problem 3\Part 1\0_Property Value Calculation.exe

Image Set:1
Area:7934.000000
Perimeter:946.000000
Euler No.:1.000000

Normalised Area:0.493040
Normalised Perimeter:1.840467
Circularity:0.111352
Aspect Ratio:0.724832

Verticality:4.613058
Horizontality:6.554071
Ratio:0.703846

Image Set:2
Area:12081.000000
Perimeter:1308.000000
Euler No.:1.000000

Normalised Area:0.514304
Normalised Perimeter:2.130293
Circularity:0.088690
Aspect Ratio:1.117241

Verticality:7.698038
Horizontality:2.268024
Ratio:3.394161

Image Set:3
Area:9205.000000
Perimeter:1044.000000
Euler No.:1.000000

Normalised Area:0.495959
Normalised Perimeter:1.912088
Circularity:0.106075
Aspect Ratio:0.882759

Verticality:8.093428
Horizontality:2.357414
Ratio:3.433180

Image Set:4
Area:7425.000000
Perimeter:1026.000000
Euler No.:1.000000

Normalised Area:0.441964
Normalised Perimeter:1.958015
Circularity:0.088501

```

Figure 1: Program Output of Geometric property for all the Training images

```

C:\Stuff\EE 569 Introduction to DIP\HW 3\Problem 3\Part 1\0_Symmetry and Spatial Moment.exe
Image:1
Moment<0,0>:7934.000000
Moment<0,1>:3909.969971
Moment<1,0>:4445.313477
Moment<1,1>:2206.182617
Centroid: <x,y>:167,147
Mid Values: <x,y>:168,148
i Shift:-1 j Shift:-1
Horizontal:51.803131
Vertical:81.330688
Image:2
Moment<0,0>:12081.000000
Moment<0,1>:6068.001465
Moment<1,0>:6718.645020
Moment<1,1>:3375.939941
Centroid: <x,y>:166,150
Mid Values: <x,y>:166,150
i Shift:0 j Shift:0
Horizontal:60.561497
Vertical:97.605583
Image:3
Moment<0,0>:9205.000000
Moment<0,1>:4613.314941
Moment<1,0>:5129.975098
Moment<1,1>:2605.250244
Centroid: <x,y>:166,149

```

Figure 2: Program Output of Spatial Moment, Shift and Symmetry for all the Training images

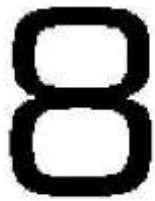
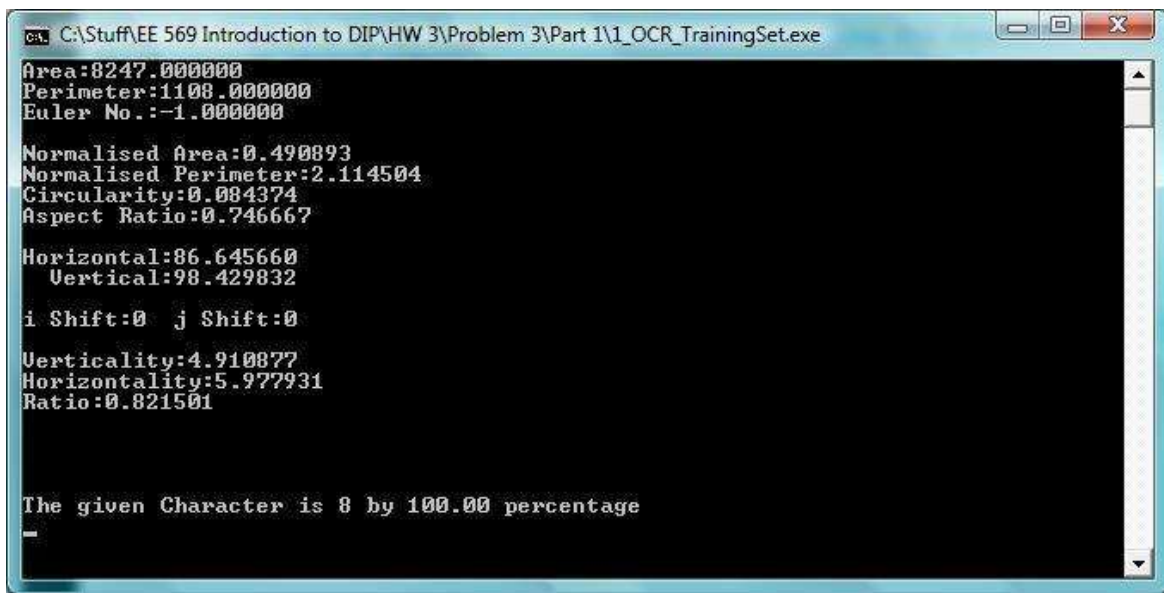


Figure 3: Training Data "training5.raw", Character "8"



```

C:\Stuff\EE 569 Introduction to DIP\HW 3\Problem 3\Part 1\1_OCR_TrainingSet.exe
Area:8247.000000
Perimeter:1108.000000
Euler No.: -1.000000

Normalised Area:0.490893
Normalised Perimeter:2.114504
Circularity:0.084374
Aspect Ratio:0.746667

Horizontal:86.645660
Vertical:98.429832

i Shift:0 j Shift:0

Verticality:4.910877
Horizontalitiy:5.977931
Ratio:0.821501

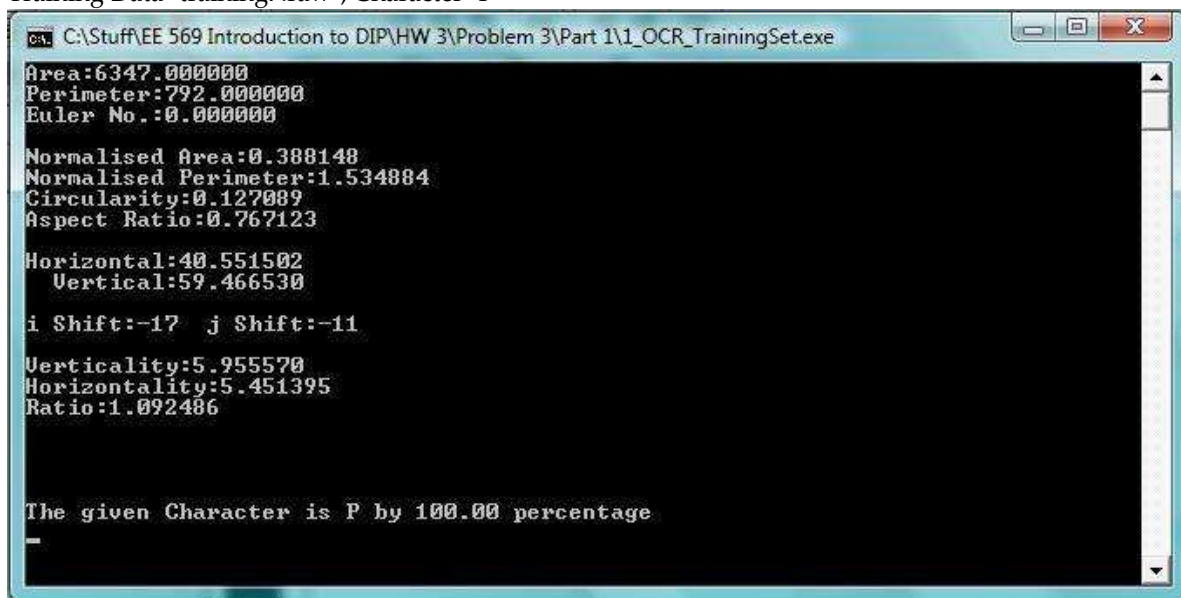
The given Character is 8 by 100.00 percentage

```

Figure 4: Program Output for "training5.raw"

IJSER

Figure 5: Training Data "training7.raw", Character "P"



```

C:\Stuff\EE 569 Introduction to DIP\HW 3\Problem 3\Part 1\1_OCR_TrainingSet.exe
Area:6347.000000
Perimeter:792.000000
Euler No.:0.000000

Normalised Area:0.388148
Normalised Perimeter:1.534884
Circularity:0.127089
Aspect Ratio:0.767123

Horizontal:40.551502
Vertical:59.466530

i Shift:-17 j Shift:-11

Verticality:5.955570
Horizontalitiy:5.451395
Ratio:1.092486

The given Character is P by 100.00 percentage

```

Figure 6: Program Output for "training7.raw"

data to deal with and also, if at least one more feature was generalised enough.

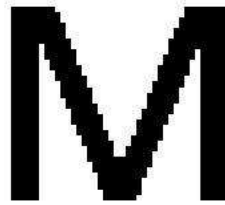


Figure 7: Training Data "training2.raw", Character "M"

```

C:\Stuff\EE 569 Introduction to DIP\HW 3\Problem 3\Part 1\1_OCR_TrainingSet.exe
Area:12081.000000
Perimeter:1308.000000
Euler No.:1.000000

Normalised Area:0.514304
Normalised Perimeter:2.130293
Circularity:0.088690
Aspect Ratio:1.117241

Horizontal:60.561497
Vertical:97.605583

i Shift:0 j Shift:0

Verticality:7.698038
Horizontality:2.268024
Ratio:3.394161

The given Character is M by 66.67 percentage
  
```

Figure 8: Program Output for "training2.raw"

#### IV. Discussion

The OCR program so developed is highly accurate in finding out the characters from the image set. Albeit the real challenge of its lies when it applied to an image set with variable font, size and orientation.

Yet, since the base classification is based on the general properties, the first one being the Euler number the system won't fail in the first instance. The cascaded weighting effects of the pseudo-general values of next three properties of symmetry, spatial moment and VH ratio would produce a decent result per-se.

The system would perform better if there was more training

proper differentiation.

## **Part(b): Optical Character Recognition, OCR Testing: Ideal Case**

### *II. Class 2*

The class 2 is then differentiated based on a) Spatial Moment (Shift) and b) Symmetry

### **I. Abstract and Motivation**

The accuracy and effectiveness of any OCR algorithm can be tested when it is used to identify images other than the ones used to train it. Our motivation here is to use OCR developed in the previous section on the testing image set and learn how effective it works, how we can improve its response better.

The use of unique, more general features in the initial point of the decision tree would bring with it some high level of accuracy. Our motivation is also to learn how we can better plot the decision tree for given the same set of features to better extract the output values.

### **II. Approach and Procedures**

The complete OCR decision tree and the property calculation is identical to the Training data set. The process just involves one extra step of iteratively inputting the entire given testing image set and churning out the values in a single instance. This just requires sting manipulation step at the file sourcing stage.

#### **File Input**

For all test image set  
StringIndex=char(IndexValue)  
Filesource<<Testing(StringIndex).raw

#### **Decision Tree**

##### *I. Euler number*

Using this we divide the character set into three classes:

Class 1, Euler no = -1: 8, since there's only one character with such an affinity we can print this one with 100% accuracy

Class 2, Euler no=0: P, R, 9

Class 3, Euler no=1: 5, M, N, S, 2, Z

The last two classes has to further undergo analysis for

It has follows:

a) Spatial Moment (Shift)

Assignment:

If RowShift<-8

$P \gg W_p = W_p + 2$

If Columnshift>0

$9 \gg W_9 = W_9 + 2$

If RowShift<-1&&Columnshift<-1

$R \gg W_r = W_r + 2$

The weights are assigned for finding how accurately the character is determined and also places **probability** in our own decision process, making the process more logical.

b) Symmetry

Assignment:

If versym<50&&horsymp<70

$P \gg W_p = W_p + 1$

I  
f

v  
e  
r  
s

y  
m  
>  
7

5  
&  
&  
h

o  
r  
s

y  
m  
p  
>  
8

5

9  
>  
>  
W  
9  
=  
W  
9  
+  
1

e  
l  
s  
e

f  
o  
r

m  
i  
d

v  
a  
l  
u  
e  
s

R  
>  
>  
W

r  
=  
W

r  
+  
1

The classification is then done by analysing the weight and finding out the percentage of accuracy from the weight also:

If  $W_p$  is large Character P

If  $W_r$  is large Character R

If  $W_9$  is large Character 9

The percentage is calculated as the ration of the weight of the found character to the total weights of all the character, this gives a measure of the OCR belief.

*III. Class 3*

The class 3 since has more character is then differentiated based on a) Spatial Moment (Shift) b)

Symmetry and VH ration(which is highly reliable for this class)

If VHratio<0.58  
 $Z \gg W_z = W_z + 1.5$

It has follows:

a) Spatial Moment (Shift)

Assignment:

If Columnshift<0  
 $n \gg W_n = W_n + 0.5$

if Rowshift<0&&Columnshift>0

$Z \gg W_z = W_z + 0.5$   
 $2 \gg W_2 = W_2 + 0.5$

The weights are assigned for finding how accurately the character is determined and also places **probability** in our own decision process, making the process more logical.

b) Symmetry

Assignment :

If horsym>85  
 $M \gg W_m = W_m + 2$

If Versym<63  
 $5 \gg W_5 = W_5 + 1$

If versym>70&&horsym>75  
 $S \gg W_s = W_s + 1.5$

Else for all other values

$N \gg W_n = W_n + 0.33$   
 $2 \gg W_2 = W_2 + 0.33$   
 $Z \gg W_z = W_z + 0.33$

c) VH Ratio

Assignment:

if VHratio>2

$N \gg W_n = W_n + 1.5$   
 $M \gg W_m = W_m + 1$

>  
>  
W  
s  
=  
W  
s  
+  
0  
.  
3  
3

The classification is then done by analysing the weight and finding out the percentage of accuracy from the weight also:

If W5 is large Character 5  
If Wm is large Character M  
If Wn is large Character N  
If Ws is large Character S  
If W2 is large Character 2  
If Wz is large Character Z

The percentage is calculated as the ration of the weight of the found character to the total weights of all the character, this gives a measure of the OCR belief.

Thus the OCR is applied on the Testing images too and the appropriate character extracted.

### III. Experimental Results

Following are the results for the problem 3(b):

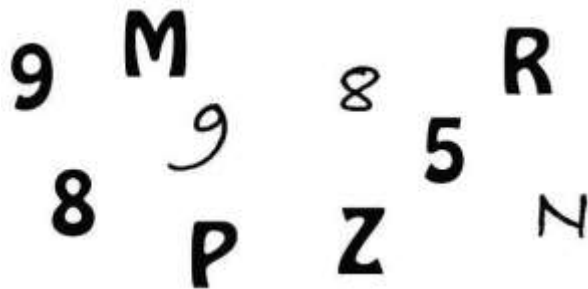


Figure 1: Complete Testing Data set



8) The high correlation of the characters from the Testing set to the training set, making the process of classification somewhat definitive

```

C:\Stuff\EE 569 Introduction to DIP\HW 3\Problem 3\Part 2\1_OCR Complete.exe
Testing Image:1
The given Character is 9 by 100.00 percentage

Testing Image:2
The given Character is 8 by 100.00 percentage

Testing Image:3
The given Character is 9 by 66.67 percentage

Testing Image:4
The given Character is Z by 58.40 percentage

Testing Image:5
The given Character is 8 by 100.00 percentage

Testing Image:6
The given Character is R by 100.00 percentage

Testing Image:7
The given Character is 5 by 57.33 percentage

Testing Image:8
The given Character is P by 100.00 percentage

Testing Image:9
The given Character is N by 50.00 percentage

Testing Image:10
The given Character is M by 66.67 percentage
  
```

Figure 2: Program Output for all the Testing Set

#### IV. Discussion

It can be noticed that the same OCR trained on the original training set gives 100% accurate result for the Testing set also. The major cause for this is

- 5) The use general (Euler) and pseudo general (Symmetry, VH Ratio) properties in the design of the OCR
- 6) The way the decision tree branches out, first comparing the most general characteristic and then on to least general one
- 7) The process of applying weights at the instances of ambiguity in pseudo-general property usage, which spreads out the probability of recognition and gives a exact belief space at each instance.

The algorithm is spot on for Class 1 having one element only and Class 2 having three elements. But for Class 3 having six elements, though it recognises the characters exactly the belief by which it recognises the character is comparatively, mainly due to the ambiguity with the non-general properties used for it. For example in the image 7 which is character 5, but due to the use of shift property and its correlative linkage to the character S, the probability drops down. For image set 9 which is character N, the use of symmetric property with high weight partially pulls in M value and reduces its belief space in that. And similarly vice-versa for the image 10 involving character M. The belief in recognising 9 for image 3 is lessened due to the use of shift property and the skewed nature of input image at some extent makes it similar looking to P character.

It can be improved either by training it to testing image also or by introducing new set of unique, general properties.

IJSER

### Part(c): Optical Character Recognition, OCR Application: License Plate Recognition

This essentially extracts out the characters in most wholesome manner, albeit it can have some inconsistencies with the localized glare and spot effect. This has to be dealt in the later section of morphological preprocessing.

b) Outer black background removal

#### I. Abstract and Motivation

The best example for the OCR application is the license number extraction from a given license plate images. It is a highly demanding problem with many different aspects of image processing involved to make even the input image to the OCR of a certain type. We have to deal with pre-processing the license plate images to extract just the region of interest with very minimal distortion and protuberance involved. The extracted license plate image might have many morphological inconsistencies which might affect the output of the OCR.

Our motivation here is to reduce the effects of all such artifacts in the OCR process by devising an efficient pre-processing technique and also, study the effects if such a snag. Our motivation here is also to measure the accuracy of the OCR algorithm and learn to build better algorithms for more general cases.

#### II. Approach and Procedures

The complete set of license images has to undergo a number of pre-processing techniques before they can be used in the OCR program to extract out the characters. The steps are:

##### *I. Character Extraction*

The characters in the license plate image are in RGB colour space and to further compound the challenge they exist within two sets of background black first, white next. So the process is done as:

a) License image binarisation

The given license plate image set is coarsely Binarised as:

```
For all image pixels
If each of RGB component > 150
BinaryData[index]=1
Else
BinaryData[index]=0
```

The outer black backdrop would produce erroneous result for the OCR program as it'd show a connected region so it had to be removed. The removal is done by the same process as the bounding box finding for the characters but only with conditional check reversed.

Inner-Boundary finding for black backdrop:

Top-Bottom and Left-Right Image search, for all image pixels If PixelValue=1

LeftBound is the lowest possible index value for column index

Top-Bottom and Right-Left Image search, for all image pixels If PixelValue=1

RightBound is the highest possible index value for column index

Left- Right and Top-Bottom Image search, for all image pixels If PixelValue=1

TopBound is the lowest possible index value for row index

Top Left- Right and Bottom-Top Image search, for all image pixels If PixelValue=1

BottomBound is the highest possible index value for row index

The character is then extruded from the bounded area and placed in a new image with full white backdrop.

## *II. Morphological Processing*

The binary images of the characters of the license plate so extracted, as mentioned earlier, are affected by the presence of tiny holes and isolated pixels, due to the effect of glare, lighting and coarse binarisation. This would give erroneous value for the Euler number which forms the first fundamental and most definitely step, has even miniscule holes is counted as a hole and isolated pixels are counted a another connected region.

So, we have to perform two morphological operations of,

Majority Black (for a set lower number of iterations) and Isolated pixel removal (single pass).

Majority Black:

For all the image pixels

Calculate the number of surrounding black pixels

If count $\geq$ 5

Then turn the current pixel to black value

Isolated Pixel Removal:

For all the image pixels

Calculate the number of surrounding black pixels

If count=0

Erase the current pixel

If Columnshift>0

9>>W9=W9+2

Once we are done with the pre-processing steps, the image sets are ready to be OCR recognised.

The complete OCR decision tree and the property calculation are identical to the Training data set. The process just involves one extra step of iteratively inputting the entire given processed image set and churning out the values in a single instance. This just requires sting manipulation step at the file sourcing stage.

### File Input

For all test image set

StringIndex=char(IndexValue)

Filesource<<LicenseImage(StringIndex).raw

### Decision Tree

#### *I. Euler number*

Using this we divide the character set into three classes:

Class 1, Euler no = -1: 8, since there's only one character with such an affinity we can print this one with 100% accuracy

Class 2, Euler no=0: P, R, 9

Class 3, Euler no=1: 5, M, N, S, 2, Z

The last two classes has to further undergo analysis for proper differentiation.

#### *II. Class 2*

The class 2 is then differentiated based on a) Spatial Moment (Shift) and b) Symmetry

It has follows:

a) Spatial Moment (Shift)

Assignment:

If RowShift<-8

P>>Wp=Wp+2

+

1

e

l

s

e

f

o

r

m

i

d

v

a

l

u

e

s

R

>

>

W

r

=

W

r

+

1

The classification is then done by analysing the weight and finding out the percentage of accuracy from the weight also:

If  $W_p$  is large Character P  
 If  $W_r$  is large Character R  
 If  $W_9$  is large Character 9

The percentage is calculated as the ration of the weight of the found character to the total weights of all the character, this gives a measure of the OCR belief.

### III. Class 3

The class 3 since has more character is then differentiated based on a) Spatial Moment (Shift) b) Symmetry and VH ration(which is highly reliable for this class)

If RowShift<-1&&Columnshift<-1  
 $R \gg W_r = W_r + 2$

The weights are assigned for finding how accurately the character is determined and also places **probability** in our own decision process, making the process more logical.

b) Symmetry

Assignment:

If  $versym < 50$  &  $horsym < 70$   
 $P \gg W_p = W_p + 1$

I

f

v

e

r

s

y

m

>

7

5

&

&

h

o

r

s

y

m

p

>

8

5

9

>

>

W

9

=

W

9

It has follows:

a) Spatial Moment (Shift)

Assignment:

If Columnshift<0  
 $n > W_n = W_n + 0.5$

if Rowshift<0&&Columnshift>0

$Z > W_z = W_z + 0.5$

IJSER

$$2 > W_2 = W_2 + 0.5$$

The weights are assigned for finding how accurately the character is determined and also places **probability** in our own decision process, making the process more logical.

b) Symmetry

Assignment:

If  $h_{orsym} > 85$

$$M > W_m = W_m + 2$$

If  $V_{ersym} < 63$

$$5 > W_5 = W_5 + 1$$

If  $versym > 70$  &  $h_{orsym} > 75$

$$S > W_s = W_s + 1.5$$

Else for all other values

$$N > W_n = W_n + 0.33$$

$$2 > W_2 = W_2 + 0.33$$

$$Z > W_z = W_z + 0.33$$

c) VH Ratio

Assignment:

if  $VH_{ratio} > 2$

$$N > W_n = W_n + 1.5$$

$$M > W_m = W_m + 1$$

If  $VH_{ratio} < 0.58$

$$Z > W_z = W_z + 1.5$$

E  
l  
s  
e

f  
o  
r

o  
t  
h  
e  
r

v  
a  
l  
u  
e  
s

2  
>  
>  
W  
2  
=  
W  
2  
+0

.  
6  
6  
5  
>  
>  
W

5  
=  
W  
5  
+  
0  
.  
3  
3

S  
>  
>  
W  
s  
=  
W

s  
+  
0  
.  
3



3

If  $W_n$  is large Character N

If  $W_s$  is large Character S

If  $W_2$  is large Character 2

If  $W_z$  is large Character Z

The percentage is calculated as the ration of the weight of the found character to the total weights of all the character, this gives a measure of the OCR belief.

Thus the OCR is applied on the processed license plate images and the appropriate character extracted.

### III. Experimental Results

Following are the results for the problem 3(c):



Figure 1: Given License Images



Figure 2: Character Extracted Images



Figure 3: Morphologically Corrected Images (Hole Filling and Isolated Pixel Removal)

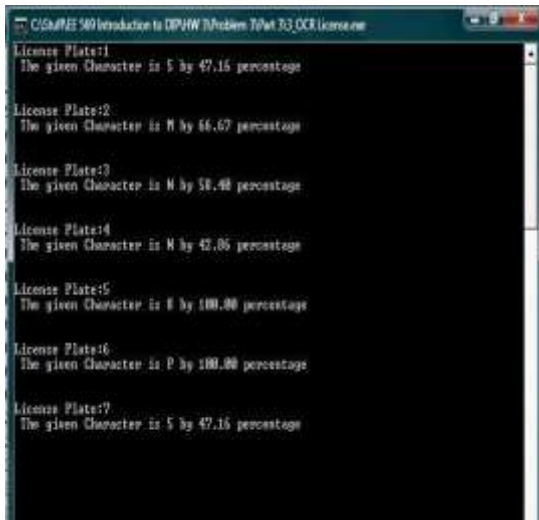


Figure 4: Program Output for all the License Image set

#### IV. Discussion

The application of the OCR program without the Morphological processing outputs erroneous values, mainly due to the effect of disturbed Euler number value so calculated based on the miniscule holes and isolated pixels. As the Euler number forms the core base of the decision tree and is the most general of all the features, the removal all such artifacts is highly important. The pre-processing step is at large independent of the input image, but the coarse quantisation might create a snag in this and needs to be improved upon.

Though, the pre-processing steps are absolutely necessary, but at some instance even they might lead to change in the character parameter so much that erratic outputs are generated for it. For example, the Majority black if overdone would thicken the character and might also remove its interior original hole, which affects how the character is seen in OCR algorithm and affects the output drastically. So, a cautionary approach to pre-processing is necessitated.

The given OCR algorithm accurately identified 5 out of 7 license character images, but fails for 2 of them, Image 4 and 6. The image 4 which is character S is read to be character N with 42.86% belief, this is mainly due to the fact that symmetry and VH ration for this given S character is high corresponding to a N (though not fully, that is why belief is

so low). The problem is hidden in the training set character S, which was less symmetrical and had average VH ratio. So, if we can train the OCR for current set, the recognition is bound to improve. The image 6 which is character 9 is read as character P with whooping 100% belief. This due to the fact that 9 character in the license plate is very less symmetrical about any axis like P from trained image, whereas 9 from the training image was the most symmetrical of all in its Class. Also, the centre of gravity is heavy-tailed towards left prominent characteristic of P, which should have been top-right has per the training image set 9.

So, it can be deduced with an absolute certainty that the training of the OCR to character sets from the license plate would give for sure an 100% accuracy.

## Part(d): Optical Character Recognition, OCR Application: Real-Life Case

### I. Abstract and Motivation

OCR application for the real-life case; even the simple ones involving license plate recognition is herculean challenge. In the real-life case the images really highly distorted, noise affected and the resolution very low. OCR requires the input images to it to be perfect with no artifacts depicting only the character to be recognised and it's been trained for. So, the task basically spurs the entire domain of the image processing and is also dependent on the assumptions made intuitively on our part to simplify the problem structure.

Our motivation here is to try recognising the license plate image of one such a real-life case and devising efficient techniques for the process. Also, our motivation is to study how robust is our OCR algorithm, and the algorithm for the complete OCR application as such.

### II. Approach and Procedures

The complete process can be summarised in three steps:

- 1) Enhance, Binarize and Segment the license plate image
- 2) Apply morphological processing to remove artifacts in each segment
- 3) Implement OCR algorithm for each such processed image and identify the character

The image processing task is easier when the image is dealt as grayscale rather than in colour space. It gives the computational efficiency and also prevents any snag due to colour channel inconsistency.

#### *I. Grayscale Conversion: Intensity Computation*

The grayscale image can be derived from the given RGB colour image by just computing the Intensity part of the HSI model and storing the Intensity value as the output image.

For all image pixels

$\text{Intensity}[] = (\text{PixelRed}[] + \text{PixelGreen}[] + \text{PixelBlue}[]) / 3$

File Store >> Intensity Value

#### *II. Symbol Extraction, Step 1: Texture Segmentation*

It is imperative that we extract just the license number images from the plate. The simple process of thresholding or even windowed adaptive thresholding won't work that efficiently, because of the presence of shadows and other artifacts. Also, the symbol grayscale is highly fluctuating and is similar to the license plate border parts.

So the best possible technique for this is texture segmentation. The texture generally as such is not that evident in any part of the license plate image, so we just use the four of the laws filter here: 1)  $L5L5^T$ , 2)  $L5E5^T$ , 3)  $E5L5^T$  and 4)  $E5E5^T$ , which would very much define the intensity based and edge based texture present in the license image.

**Assumptions:** *The texture present in the image consists of only the level, edge and their combination based texture.  $K=4$ , that is there's four types of textures in the license plate image: a) The Symbol, b) the plate boundary, c) white space and d) the shadow part.*

The centroid position for such  $K=4$  is chosen at the point of each of such assumed textures and the K-mean segmentation technique is applied and iterated till the convergence.

We obtain, the segmented image as shown in the figure 3; though the symbol are still not that clearly demarcated, the difference between the symbol points and the background as a whole is now a *constant*.

### III. Symbol Extraction, Step 2: Coarse Binarisation

Now we that we have clear distinction between the symbol and major part of the background, we can try extract the symbol collage as such. This done by blanket thresholding, which is again a coarse binarisation. The most backdrop in the texture segmented image in of grayscale value 0 and the rest has one the three other label grayscale value. The coarse binarisation is done by changing the backdrop to white and all other image part to black.

For all image pixels

I  
f

P  
i  
x  
e  
l  
V  
a  
l  
u  
e

[  
]  
=  
0  
  
B  
i  
n  
a  
r  
y  
I  
m  
a  
g  
e  
[  
]  
=  
1  
E  
l  
s  
e  
f  
o  
r  
  
a  
l  
l  
  
t  
h  
e  
  
r  
e  
s  
t  
  
B  
i  
n  
a  
r  
y  
I  
m  
a  
g  
e  
[

]
   
=
   
0

Now we get a binary image which has some resemblance to the symbol to be extracted, as shown in the figure 4.

#### *IV. Symbol Extraction, Step 3: Image rotation*

The obtained license plate image is rotated to a certain extent wrt to the reference coordinates. For proper symbol segmentation, it necessary that they are aligned wrt to the reference axis. So, depending on the angle of rotation the image is offset by rotating in the opposite direction and thus aligning.

***Assumption:*** *The image is found to be rotated by an angle of 8degrees(found manually and by trial and error approach, not by program)*

The geometric transformation of rotation is then carried out with implicit bilinear interpolation and the aligned image is thus generated as shown in the figure 5.

IJSER

#### *V. Symbol Extraction, Step 4: Pruning and Border correction*

The symbols are found to be located only within the range of (165,460) height-wise and (80,1190) width-wise. So a new image is created of height, 295 and width, 1110 which contains only the symbol data to a large extent as shown in the figure 6.

For all image pixels within the range  
 $\text{NewImage}[\text{translatedIndex}] = \text{OldImage}[]$

#### *VI. Symbol Extraction, Step 5: Pruning and Border correction*

Each of the symbols in the pruned image was found to occupy equal space in the range of 160.

**Assumption:** *The symbols are only 160 wide and are equally spaced in the license plate*

So each of the symbols is cut and placed in new image of size (300X300) with a white backdrop. The symbols are placed in the large white space to allow easier detection and manipulation.

For all the symbols; 7 iteration  
 For all the image pixel within the bound  
 $\text{SegmentedSymbol}[\text{offsetindex}] = \text{ImageData}[]$   
 And the symbols are stored in a sequential manner.

#### *VII. Morphological Processing*

The segmented image has many erratic holes and stray pixels. We need to fill out the non-symbolic holes and remove stray pixels to get exact Euler number for having clear first step of decision making during OCR.

The holes (and some isolated pixel blocks) within each symbol are somewhat bigger and normal morphological process like majority black won't fill them (or remove them). So to reduce the size of the holes and still retain the image characteristic, we scale down image by a factor of 5, creating segments of (60X60) size and has really small holes that can be removed by single pass of Majority Black

operation.

So we apply two operations of Down-Scaling and Majority Black operation.

The output of the operation creates an almost symbol resembling image sets, as shown in the figure 9.

#### *VIII. OCR implementation*

The OCR algorithm is then executed on these morphologically processed image segments. And the license plate values are obtained.

Thus OCR implementation for a real-life scenario was implemented and results obtained.

### III. Experimental Results

Following are the results for the problem 3(d):



Figure 1: Actual License Plate Image (1200x500)



Figure 2: Gray Scale License Plate by Intensity Calculation (1200x500)

Figure 5: Rotated License Plate Image (1200x500)



Figure 3: Texture Segmented License Plate Image, K=4  
(1200x500)



Figure 4: Coarse Binarised License Plate Image (1200x500)







Figure 6: Pruned and Border Corrected License Plate Image (1110X295)



Figure 7: Segmented License Plate Image (300X300)



Figure 8: Down-Scaled License Plate Image (Factor of 5) (60X60)



Figure 9: Morphologically Corrected License Plate Image (Majority Black) (60X60)

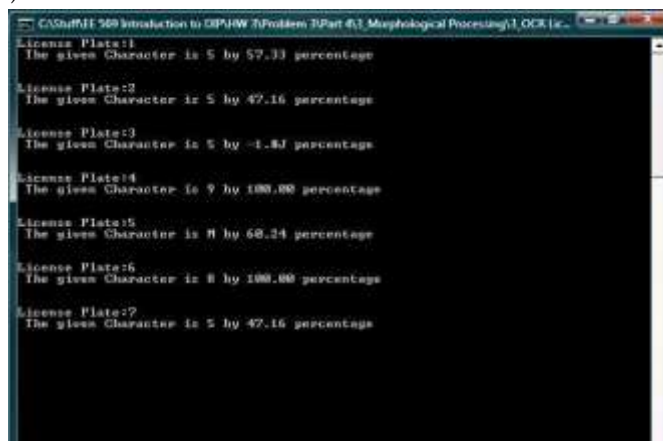


Figure 10: OCR Program Output

#### IV. Discussion

As it can be seen from the figure 10, the OCR program accurately identifies only 2 of the 7 given symbols on the license plate. It is due to

- 1) For image 2, the given character is P but the program detects it as 5 because there's a region of stray pixel block which shows as connected region to the program and the Euler number so generated is -1 instead of 0 for P. And depending on closest approximation in Class 3, 5 is selected as the output candidate. The same effect is prominent in the image 3 which has character Z, but due to morphological discrepancies it gives out the Euler value of -2 and so the output is given out as 5, which was the remnant of the last output for image 2.
- 2) For image 4, 5, 7 though the Euler value for it is detected correctly, the pre-processing steps, especially

#### CONCLUSION & FUTURE SCOPE OF WORK

The OCR Signal Generation Software bundle process the required data. The devised OCR is robust to an extent, but since it is trained on a limited set of data its accuracy is also limited. Albeit, by the use of better pre-processing techniques that remove the isolated pixel block, smooth out the symbol edge and proper characterize the symbols, we can still very effectively have a good OCR determination for the license plate image.

Only one simple feature of exactly extracting out just binary image of symbol from the license plate would markedly improve the performance of the OCR for the real life case. Better techniques I believe would be a cascade of steps, Noise removal >> Edge enhancement >> Texture Segmentation based on more features and better initial points >> Edge detection and Thinning >> Bound Filling >> Segmentation >> Edge Smoothing >> Basic Morphological Process >> and then OCR.

Modified OCR Signal generation can be implemented using new algorithm. Test modules can be done with other applications. Exact code and test modules can be used in real production and testing with the enhanced tool.

majority black and segmentation, has left an indelible mark on the symbol. Due to the way the decision tree was built placing weights on symmetrical and spatial moment aspect, the thickened symbols are erroneously calculated as their class counterparts which had high symmetry and low spatial shift based on the training image set.

The devised OCR is robust to an extent, but since it is trained on a limited set of data its accuracy is also limited. Albeit, by the use of better pre-processing techniques that remove the isolated pixel block, smooth out the symbol edge and properly characterize the symbols, we can still very effectively have a good OCR determination for the license plate image.

Only one simple feature of exactly extracting out just binary image of symbol from the license plate would markedly improve the performance of the OCR for the real life case. Better techniques I believe would be a cascade of steps, Noise removal >> Edge enhancement >> Texture Segmentation based on more features and better initial points >> Edge detection and Thinning >> Bound Filling >> Segmentation >> Edge Smoothing >> Basic Morphological Process >> and then OCR.

#### REFERENCE

- [1] Digital Image Processing 4<sup>th</sup> Edition, William K. Pratt, Wiley
- [2] EE 569 Lecture Notes 2,3,5,6; Professor C. Jay Kuo
- [3] EE 569 Discussion Slides 4,5,6; Hang Yuan, Sanjay Purushotham and Sachin Chachada
- [4] A. Buades, B. Coll, J.M. Morel, "A non local algorithm for image denoising", IEEE International Conference on Computer Vision and Pattern Recognition, CVPR 2005.vol.2; pp: 60-65
- [5] nlm.cpp, source code, EE569 homework1 materials, uscdcn.net
- [6] CannyEdgeDetection.cpp, source code, EE569 homework2 materials, uscdcn.net
- [7] [http://en.wikipedia.org/wiki/Canny\\_edge\\_detector](http://en.wikipedia.org/wiki/Canny_edge_detector)
- [8] Digital Image Processing 4<sup>th</sup> Edition, William K. Pratt, Wiley
- [9] EE 569 Lecture Notes 7,8,10; Professor C. Jay Kuo
- [10] EE 569 Discussion Slides 7,8,9,10; Jiangyang Zhang, Hang Yuan and Sanjay Purushotham
- [11] <http://www.blackpawn.com/texts/pointinpoly/default.html>