1.2 HISTORY

Many techniques of digital image processing were developed in ,1960s at the Jet Propulsion

Laboratory, Massachusetts Institute of Technology, Bell Laboratories, University of Maryland.

The cost of processing of image was very high. But that changed in 1970s, when digital image

processing grow rapidly as cheaper computer and hardware became available. Images then

could be processed in real time. With the development of fast computers available in

200s,digital image processing has become one of the most common form of image processing. It

is used because it is not only the most versatile method, but also the cheapest

***INTRODUCTION***

DEFINATION - Digital image processing is concerned with processing of an image. Image processing is a method to perform operations on images like enhancing images, extracting text from image, detecting edge of image and many other operations. In digital image processing we take an image and convert that image in different forms. Like if we take colour image, we can convert it into grey image. In this both the input and output is an image. Usually Image Processing system includes treating images as two-dimensional signals while applying already set signal processing methods to them. Today, it is rapidly growing technology. It forms core research area within engineering and computer science disciplines too. Image processing has its wide applications in robotics, machine learning, neural networking, signal processing, medical field, graphics and animations and in many other fields.

***HISTORY***

Many techniques of digital image processing were developed in ,1960s at the Jet Propulsion Laboratory, Massachusetts Institute of Technology, Bell Laboratories, University of Maryland. The cost of processing of image was very high. But that changed in 1970s, when digital image processing grows rapidly as cheaper computer and hardware became available. Images then could be processed in real time. With the development of fast computers available in20s, digital image processing has become one of the most common form of image processing. It is used because it is not only the most versatile method, but also the cheapest.

***APPLICATIONS***

Almost in every field, digital image processing puts a live effect on things and is growing with time to time and with new technologies.

1. Image sharpening and restoration-It is the process in which we can modify the image. We can convert the color image to grey image, sharpening, enhancement of the image, detecting edges, and recognition of images.

2. Medical field-Now a days if we have brain tumor through the image processing the tumor is detect that where the tumor is. Also, it is used to detect any kind of cancer. Xray imaging, medical CT Scan, UV imaging depends on the functioning of digital image processing.

3. Robot -Vision-There are several robotic machines which work on this technique. Through this technique robots find their ways. Like they can detect the hurdle and line follower robot.

4. Pattern- recognition-It involves study of image- processing. It is also combined with the artificial intelligence such that computer-aided diagnosis, handwriting- recognition and images- recognition can be easily implemented.

5. Video processing-The collection of frames and pictures are arranged in such a way that movement of pictures become faster. It involves frame rate, motion detection, reduction of noise and color space conversion etc.

***HOW TO CONVERT An RGB IMAGE INTO GRAYSCALE IMAGE***

***Average method***

Average method is the simplest one. You just have to take the average of three colours. Since it’s an RGB image, so it means that you have add r with g with b and then divide it by 3 to get your desired grayscale image.

It’s done in this way.

Grayscale = (R + G + B / 3)

For example:



If you have a color image like the image shown above and you want to convert it into grayscale using average method. The following result would appear.



Explanation

There is one thing to be sure, that something happens to the original works. It means that our average method works. But the results were not as expected. We wanted to convert the image into a grayscale, but this turned out to be a rather black image.

Problem

This problem arises due to the fact, that we take average of the three colours. Since the three different colours have three different wavelength and have their own contribution in the formation of image, so we have to take average according to their contribution, not done it averagely using average method. Right now, what we are doing is this,

33% of Red, 33% of Green, 33% of Blue

We are taking 33% of each, that means, each of the portion has same contribution in the image. But in reality, that’s not the case. The solution to this has been given by luminosity method.

***Weighted method or luminosity method***

You have seen the problem that occur in the average method. Weighted method has a solution to that problem. Since red color has more wavelength of all the three colours, and green is the color that has not only less wavelength then red color but also green is the color that gives more soothing effect to the eyes.

It means that we have to decrease the contribution of red color, and increase the contribution of the green color, and put blue color contribution in between these two.

So, the new equation that form is:

New grayscale image = ((0.3 \* R) + (0.59 \* G) + (0.11 \* B)).

According to this equation, Red has contributed 30%, Green has contributed 59% which is greater in all three colours and Blue has contributed 11%.

Applying this equation to the image, we get this

Original Image:



Grayscale Image:



Explanation

As you can see here, that the image has now been properly converted to grayscale using weighted method. As compare to the result of average method, this image is brighter.

***HOW TO CONVERT A GRAYSCLAE IMAGE INTO BINARY IMAGE***

Thresholding is the simplest method of image segmentation and the most common way to convert a grayscale image to a binary image.

In thresholding, we select a threshold value and then all the gravy level value which is below the selected threshold value is classified as 0(black i.e. background) and all the gravy level which is equal to or greater than the threshold value is classified as 1(white i.e. foreground).

for(int I =0;i<imgSize;i++)

{

//buffer[i] = (buffer[i]>THRESHOLD)? WHITE : BLACK;

if (buffer[i]>THRESHOLD)

{

buffer[i] = WHITE ;

} else

buffer[i] = BLACK ;

}

HERE VALUE OF THRESHOLD IS 120.

BLACK = 0.

WHITE = 255.

Original image after converting into a grayscale image

***HOW TO ADD TWO GRAYSCALE IMAGES***

ADDITION OPERATION IS APPLIED ON PIXEL BY PIXEL BASIS. MEANING THE OPERATION IS APPLIED TO EACH PIXEL IN THE IMAGE X + Y = Z

X is a 2d array

Y is either a 2d array of same size or a constant value

Z is also a 2d array of same size as that of X

IF Y IS A CONSTANT THAN IF Y IS +ve THAN IMAGE WILL EXPERIENCE AN INCREASE IN BRIGHTNESS

AND IF Y IS -ve THAN IMAGE WILL EXPERIENCE AN DECREASE IN BRIGHTNESS

AND IF Y IS A 2D ARRAY THAN CONTENT OF X AND Y WILL BLEND IN Z.

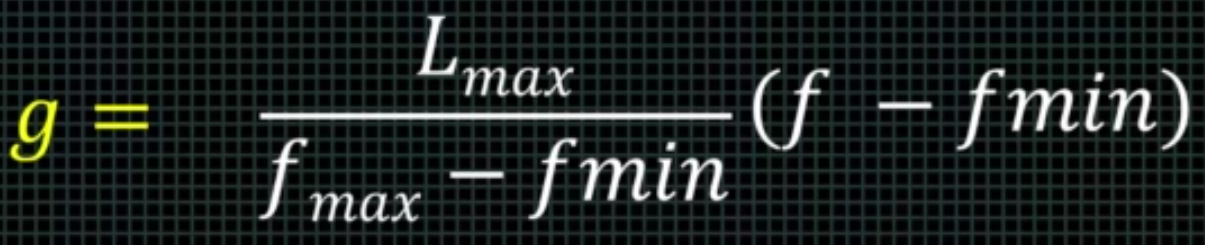
NOTE -: WE SHOULD BE CAREFULL WHILE ADDING, THAT THE VALUE OF Z SHOULD NOT MORE THAN 255 AND TO OVERCOME THIS PROBLEM WE CAN EITHER WRITE A IF STATEMENT THAT

IF(Z>255)

THEN Z= 255

OR WE CAN USE THE METHOD OF NORMALIZATION. WHICH IS MUCH BETTER APPROACH

WE USE THE FORMULA





FIRST IMAGE SECOND IMAGE



RESULT

***HOW TO SUBTRACT TWO GRAYSCALE IMAGE***

SUBTRACTION OPERATION IS APPLIED ON PIXEL BY PIXEL BASIS. MEANING THE OPERATION IS APPLIED TO EACH PIXEL IN THE IMAGE X - Y = Z

X is a 2d array

Y is either a 2d array of same size or a constant value

Z is also a 2d array of same size as that of X

IF Y IS A CONSTANT THAN IF Y IS -ve THAN IMAGE WILL EXPERIENCE AN INCREASE IN BRIGHTNESS

AND IF Y IS +ve THAN IMAGE WILL EXPERIENCE AN DECREASE IN BRIGHTNESS

AND IF Y IS A 2D ARRAY THAN CONTENT OF X AND Y WILL BLEND IN Z.

NOTE -: WE SHOULD BE CAREFULL WHILE SUBTRACTING, THAT THE VALUE OF Z SHOULD NOT BE LESS THAN 0 AND TO OVERCOME THIS PROBLEM WE CAN EITHER WRITE A IF STATEMENT THAT

IF(Z<0)

Z = 0

OR WE SHOULD TAKE ABSOLUTE VALUE OF Z.



FIRST IMAGE SECOND IMAGE



RESULT

***HOW TO ENLARGE AN IMAGE***

TO ENLARGE AN IMAGE JUST KEEP VALUE OF S1 AND S2 GREATER THAN 1

for(int x =0;x<width;x++)

{

for(int y = 0;y<height;y++)

{

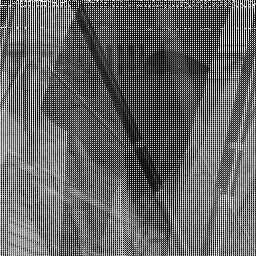
x1 = x\*s1 ;

y1 = y\*s2 ;

out\_buffer[x1][y1] = buf[x][y];

}

}

ORIGINAL IMAGE IMAGE AFTER INLARGMENT

***HOW TO SHRINK AN IMAGE***

TO SHRINK AN IMAGE JUST KEEP VALUE OF S1 AND S2 BETWEEN 0 AND 1

for(int x =0;x<width;x++)

{

for(int y = 0;y<height;y++)

{

x1 = x\*s1 ;

y1 = y\*s2 ;

out\_buffer[x1][y1] = buf[x][y];

}

}



ORIGINAL IMAGE IMAGE AFTER SHRINKING

***HOW TO CROP AN IMAGE***

For(int x =h;x<width;x++)

{

for(int y = k;y<height;y++)

{

x1 = x ;

y1 = y ;

out\_buffer[x1][y1] = buf[x][y];

} }

ORIGINAL IMAGE IMAGE AFTER CROPING

***REFLECTION OF IMAGE***

Right before calling the function reflectImage(int flag) , we take the input of flag to check whether user wants to reflect the image horizontally or vertically.

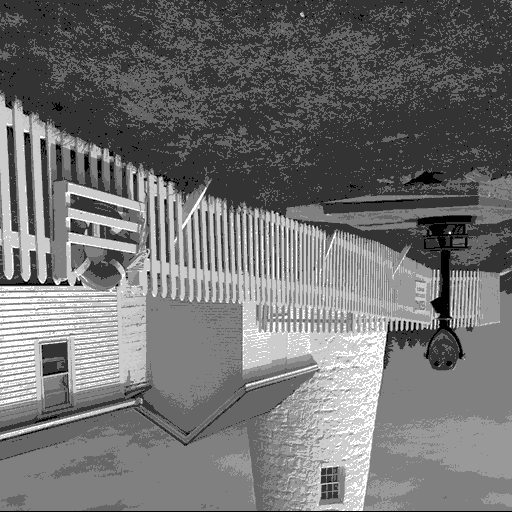
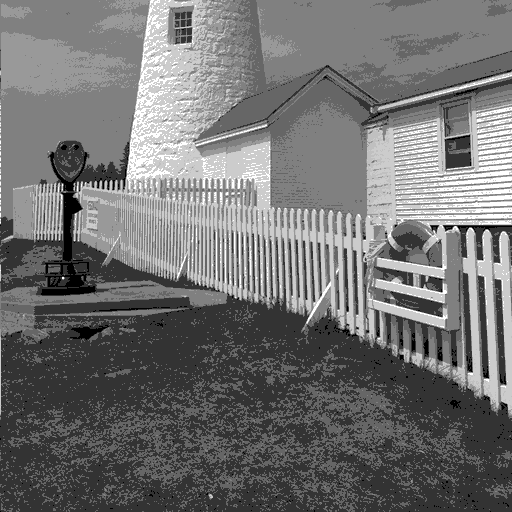
In the function same process as of reading the image is followed .

If flag==1 , we use following constraints to reflect the image vertically :

x2=height - x1 & y2=y1;

If flag==0 , we use the following constraints to reflect the image horizantally :

x2=x1 & y2=height -y1;

original image vertically reflected horizantally reflected

**DETAILS OF IMAGE**

Inside the function getImageDetail(int moRows,int noCols,int maxVal) , first we read a file as usual then we read the values of height(no of rows) and width(no of columns) from the header of the file by using :

noCols=\*(int \*)&header[18]; noRows=\*(int \*)&header[22];

then we took out the maximum value of the pixel in the image by using a simple nested for loop :

for(int i=0;i<noCols;i++)

for(int j=0;j<noRows;j++)

{

if((int)buff[i][j] > maxVal)

maxVal=(int)buff[i][j];

}

and finally printed all the values and closed the file .

***COPY IMAGE***

In the function writeImage() , we simply read the file by using fread() & saved all the data such as header , colortable and buffer & wrote all of that in another file using fwrite() .

***IMAGE TRANSLATION***

In the function imageTranslate() , we just simply moved the image in X-Direction by some amount "t" . And for doing so we use the nested for() loop and performed the following function :

for(int x =0;x<width;x++)

{

for(int y =0;y<height;y++)

{

if((y+t)<512)

out\_buffer[x][y+t] = buffer[x][y];

else

out\_buffer[x][512-(y+t)]=buffer[x][y];

}

}

And finally wrote the changed value of buffer into a new file.

original image translated image by some amount "t"

***NEGATIVE OF AN IMAGE***

In the function imageNegative() , we just simply substracted each pixel value by 255 to get the -ve image as shown below :

for(i=0;i<width;i++)

{

for(j=0;j<height;j++)

{

buffOut[i][j]=255-buffIn[i][j];

}

original image negative image

***IMAGE ENHANCEMENT FILTERS***

Under this we used two function imageBlur() & imageSepia() which respectively blurred and added sepia filter to the image.

IMAGE BLUR

In the function imageBlur() , we blurred the image by first reading the RGB values of a pixel and then used algorithm mentioned below :

float kernel[3][3]={{1.0/9.0,1.0/9.0,1.0/9.0} , {1.0/9.0,1.0/9.0,1.0/9.0} , {1.0/9.0,1.0/9.0,1.0/9.0}};

for(int x=1;x<height-1;x++)

{

for(int y=1;y<width-1;y++)

{

float sum0=0.0;

float sum1=0.0;

float sum2=0.0;

for(int i=-1;i<=1;i++)

{

for(int j=-1;j<=1;j++)

{

sum0 = sum0 + (float)kernel[i+1][j+1] \* buffIn[(x+i)\*width+(y+j)][0];

sum1 = sum1 + (float)kernel[i+1][j+1] \* buffIn[(x+i)\*width+(y+j)][1];

sum2 = sum2 + (float)kernel[i+1][j+1] \* buffIn[(x+i)\*width+(y+j)][2];

}

}

buffOut[(x)\*width+(y)][0] = sum0;

buffOut[(x)\*width+(y)][1] = sum1;

buffOut[(x)\*width+(y)][2] = sum2;

}

}

Finally we wrote the new buffer values to our output file .

original image blurred image

SEPIA IMAGE

In the function imageSepia() , we used the following algorithm :

for(int i=0;i<imgSize;i++)

{

r=g=b=0;

buffIn[i][0]=getc(fin); //red

buffIn[i][1]=getc(fin); //green

buffIn[i][2]=getc(fin); //blue

r=(buffIn[i][0]\*0.393) + (buffIn[i][1]\*0.769) + (buffIn[i][2]\*0.189);

g=(buffIn[i][0]\*0.349) + (buffIn[i][1]\*0.686) + (buffIn[i][2]\*0.168);

b=(buffIn[i][0]\*0.372) + (buffIn[i][1]\*0.534) + (buffIn[i][2]\*0.131);

if(r>MAX\_PIXEL)

r=MAX\_PIXEL;

if(g>MAX\_PIXEL)

g=MAX\_PIXEL;

if(b>MAX\_PIXEL)

b=MAX\_PIXEL;

putc(b,fout);

putc(g,fout);

putc(r,fout);

original image after using sepia filter

***GET PIXEL VALUE AT (r,c)***

In the function getPixelValue() , we first read the file and then we read the values of all the pixels in the image and stored them in the array buffer[][] .

Then we took the input of r & c and printed the pixel value at that positiion.

***SET PIXEL VALUE AT (r,c)***

In the function setPixelValue() , we first read the file and then we took the input of the values of r & c and then we took the input of the new pixel value at (r.c) [which should be always between 0 to 255 because we used a grey image and the pixel value of shades of grey varies in this range] . Then we wrote the new pixel value in the file.

***SUBMITTED BY – VINAYAK CHACHRA***

***ROLLNO – 191210061***

***SUBMITTED TO – Dr. Rishav Singh***