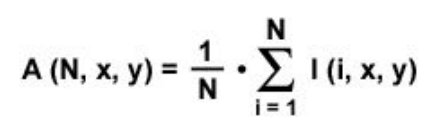
**Practical No 14.C**

**Implementation of histogram equalization - Image averaging.**

**Aim: Write a program to implement a histogram equalization using Image averaging algorithm.**

**Theory:**

Image averaging is a digital image processing technique that is often employed to enhance video images that have been corrupted by random noise. The algorithm operates by computing an average or arithmetic mean of the intensity values for each pixel position in a set of captured images from the same scene or viewfield. Each corrupted image has a stable signal component and a random noise component. In the averaging process, the signal component of the image remains the same, but the noise component differs from one image frame to another. Because the noise is random, it tends to cancel during the summation. When the averaged image is computed, the image signal component has a stronger influence over the summation than does the noise component. The result is an enhanced signal component, while the noise component tends to be reduced by a factor approximately equal to the square root of the number of images averaged. where the sum is taken over I (i, x, y), which is the pixel intensity value of the ith frame from the N set of image frames. This form of the algorithm requires that all of the N image frames be available simultaneously,

which typically requires a considerable amount of computer memory (RAM). An important application of image averaging is in the field of astronomy, where imaging with very low light levels is routing, causing sensor noise frequently to render single images virtually which is useless for analysis.

**Conclusion: We have implemented histogram equalization using Image averaging algorithm.**

**Code:**

#include<iostream.h>

#include<fstream.h>

#include<conio.h>

#include<string.h>

#include<math.h>

struct pix

{

unsigned char b,g,r;

}pixel1,pixel2;

int L=255;

char Header[54];

int c;

ifstream in1,in2;

ofstream out,out1;

class sub

{

public:

void process()

{

char infile1[]="cat1.bmp";

char infile2[]="cat3.bmp";

char outfile[]="cat-avg.bmp";

char imdata[]="imdata.dat";

in1.open(infile1,ios::in|ios::binary);

in2.open (infile2,ios::in|ios::binary);

in1.read(( char\*)(&Header),sizeof(Header));

in2.read(( char\*)(&Header),sizeof(Header));

out.open(outfile,ios::out| ios::binary);

out.write(( char\*)(&Header),sizeof(Header));

out1.open(imdata, ios::out);

while(!in1.eof())

{

pix a1,a2;

in1.read((char \*)(&pixel1),sizeof(pixel1));

a1.r=pixel1.r;

a1.g=pixel1.g;

a1.b=pixel1.g;

out1<<"ORIGINAL 1: " <<(int)pixel1.r<<" , "<<(int)pixel1.g<<" , "<<(int)pixel1.b<<endl;

in2.read((char \*)(&pixel1),sizeof(pixel1));

a2.r=pixel1.r;

a2.g=pixel1.g;

a2.b=pixel1.g;

out1<<"ORIGINAL 2: " <<(int)pixel1.r<<" , "<<(int)pixel1.g<<" , "<<(int)pixel1.b<<endl;

pixel1.r=(float)(a1.r+a2.r)/2.0;

pixel1.g=(float)(a1.g+a2.g)/2.0;

pixel1.b=(float)(a1.b+a2.b)/2.0;

out.write((char \*)(&pixel1),sizeof(pixel1));

out1<<"MODIFIFED (average) : " <<(int)pixel1.r<<" , "<<(int)pixel1.g<<" , "<<(int)pixel1.b<<endl;

}

in1.close();

in2.close();

out.close();

}

};

int main()

{

clrscr();

sub s;

s.process();

getch();

}

**Input:**

**Output:**

