Coded by RickWare

Histogram

void convert()

{

unsigned char Gray;

unsigned int histcount[256];

char string[10];

//Initialize histogram count to 0

for (int i=0;i<256;i++)

histcount[i] = 0;

//Add \*.pgm Header

fprintf(outfptr,"P5\n512 512\n255\n");

//Perform counts

Gray = fgetc(infptr);

do{

//Increment histogram for grayscale level

histcount[Gray]++;

//Read next pixel to check if we have an End of File

Gray = fgetc(infptr);

} while (!feof(infptr));

//Convert the histogram values from integers to string

for (int i=0;i<256;i++){

itoa (histcount[i],string,10);

fputs(string,outfptrh);

fputs("\n",outfptrh);

}

}

Mirror

void negative()

{

unsigned char Gray;

unsigned char mat[512][512];

int ren, col;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n512 512\n255\n");

//Read input image and store en matrix

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

Gray = fgetc(infptr);

mat[ren][col] = Gray;

}

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

//Invert columns

Gray = mat[ren][511-col];

fputc(Gray,outfptr);

}

}

50% Reduction

void negative()

{

unsigned char Gray;

unsigned char mat[512][512];

int ren, col;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n256 256\n255\n");

//Reading input image and storing in a matrix

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

Gray = fgetc(infptr);

mat[ren][col] = Gray;

}

for (ren = 0; ren < 256; ren++)

for (col = 0; col < 256; col++) {

//Invert columns

Gray = mat[ren\*2][col\*2];

fputc(Gray,outfptr);

}

}

RGB to Gray

void convert()

{

unsigned char R,G,B,Gray;

char string[10];

//Add \*.pgm Header

fprintf(outfptr,"P5\n512 512\n255\n");

//Perform conversions

R = fgetc(infptr);

do{

G = fgetc(infptr);

B = fgetc(infptr);

//Gray =(R + G + B) / 3;

Gray = B;

fputc(Gray,outfptr);

//Read next pixel to check if we have reached End of File

R = fgetc(infptr);

} while (!feof(infptr));

}

Grayscale to binary

void binary()

{

unsigned char Gray;

unsigned int Threshold = 192;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n512 512\n255\n");

//Perform conversions

Gray = fgetc(infptr);

do{

if (Gray > Threshold)

fputc(255,outfptr);

else

fputc(0,outfptr);

//Read next pixel to check if we have an End of File

Gray = fgetc(infptr);

} while (!feof(infptr));

}

Edge detection (not working)

void negative()

{

unsigned char Gray;

unsigned char mat[512][512];

unsigned char matshift[512][512];

int ren, col;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n512 512\n255\n");

//Convert input image to a matrix

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

Gray = fgetc(infptr);

mat[ren][col] = Gray;

}

//Shift entire image 1 pixel to the right

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

//Shift Matrix

if (col = 0)

matshift[ren][col] = 0;

else

matshift[ren][col-1] = mat[ren][col];

}

//Subtract shifted matrix from original

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

Gray = matshift[ren][col] - mat[ren][col];

fputc(0,outfptr);

}

}

Double Size

void negative()

{

unsigned char Gray;

unsigned char mat[1024][1024];

int ren, col;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n1024 1024\n255\n");

//Perform conversions

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

Gray = fgetc(infptr);

mat[ren\*2] [col\*2] = Gray;

mat[ren\*2+1][col\*2] = Gray;

mat[ren\*2] [col\*2+1] = Gray;

mat[ren\*2+1][col\*2+1] = Gray;

}

for (ren = 0; ren < 1024; ren++)

for (col = 0; col < 1024; col++) {

//Invert columns

Gray = mat[ren][col];

fputc(Gray,outfptr);

}

}

Histogram Stretching

void negative()

{

unsigned char Gray;

//Matrix Variables

unsigned char mat[512][512];

int ren, col;

//Maximum and Minimum pixel values

int max = 0;

int min = 255;

//Correction Factor

float ScaleFactor;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n512 512\n255\n");

//Read input image

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

//Read Pixel

Gray = fgetc(infptr);

//Store Pixel in Matrix

mat[ren][col] = Gray;

//Find lowest and highest pixel values in input image

if (Gray < min) min = Gray;

if (Gray > max) max = Gray;

}

printf("Maximum pixel value:%d\n", max);

printf("Minimum pixel value:%d\n", min);

//Correction Factor

ScaleFactor = 255.0/(max-min);

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

//Invert columns

Gray = mat[ren][col];

Gray = (Gray-min)\*ScaleFactor;

fputc(Gray,outfptr);

}

}