CRANFIELD UNIVERSITY

ASSIGNMENT – III

PROGRAMMING METHODS FOR ROBOTICS

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING

Mr. Balasekhar Chandrasekaran Kannan

S359368

M.Sc. Robotics

Academic Year : 2021 – 2022

Faculty : Dr. Irene Moulitsas

January 2022

ABSTRACT

Information possessed by the image cannot be extracted from it without processing the data. This study discusses about three important processes- Smoothing, sharpening and edge detection. Given binary ppm image is converted to P3 format and filters are applied on the image. A 10*10-pixel image is used in order to validate the result obtained. Furthermore, the importance of order of operation between smoothening and sharpening for optimal output is also studied. The C++ program gives the P3 converted file, smoothened P3, sharpened P3 and edge detection P3 when a P6 image file is inputted.

KEYWORDS : Smoothening, Sharpening, Edge detection, Image processing.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Image analysis is concerned with quantifying the data of the objects present in the scene and capture their properties such as position, orientation etc. The field of computer vision uses image analysis to obtain meaningful information from the image process it. There is an utmost need to improve the efficiency of the analysis process due to its introduction in wide variety of fields.

Any image captured will have certain level of noise and disturbance. These can distort the features that image is trying to convey. Pixel values of colour levels obtained by the image is not same as the real value. This can occur due to signal disturbances, sensor trouble, unwanted exposure, improper capturing conditions, relative movement between camera and object etc. CCD sensor malfunction is one of the major causes for addition of impulsive noise. Therefore it is important that the image has to be processed before it can be implemented for feature analysis. These steps can enhance the image by introducing contrast between similar features, compress for better computation, de-blur helps in restoration and also feature extraction (Maria Petrou, 2010).

Processing of image can be performed by using operators. These operators act like transfer function by taking in the raw image and giving us the filtered result. The operators can be linear or non-linear. Kernel convolution matrices are example of linear operators. Three filtering methods are discussed in this study (GJ Awcock, 1995).

Portable pixel map also abbreviated as PPM is an image format that details the colour image format. It is a highly complicated format for understanding and is usually preferred for writing and analysing program codes for manipulation. This format contains information such as the dimensions of the image, maximum pixel value and RGB values for every individual pixel in the image. Convolutions can be easily applied to this format for processing the image. They are kernels specific to a single operation in the spatial plane.

Sections 1.1, 1.2, 1.3 explains about the advantages and disadvantages of smoothing, sharpening and edge detection. Section 1.4 details the advantages of implementing these codes in c++. Section 2 covers the methodology used for successful running of the program with three sub-divisions to break down the code part by part. Section 3 validates the process implemented by the codes explained in Section 2.

1.1. SMOOTHING

The process of smoothening or blurring the image, or to create a less pixelated image by removing noise. Additive noise is present in such a way that the form of the original image is still preserved. These can cause zero mean areas and destroy information (Forsyth, 2001).

The smoothing method use in our study is also called as a convolution method since a 3*3 matrix is passed on over the entire image to obtain the result. Average or median of the surrounding pixel is generated from the convolution matrix. This method of getting average over a 3*3 matrix is also called as moving window average.

Gaussian noise is where pixel value is changed by a level that follows a gaussian distribution. This discussion deals with linear implementation of nonlinear methods such as the Bilateral filter, which is based on the average mean property of gaussian noise. Smoothing an image eliminates this gaussian difference and allows us to better grasp the erotic changing values.

This method can also be used to extract objects, remove dead pixels and fill in undesired gaps. The BF approach uses a smoothing factor of 25% for all of its nearby neighbours. The smoothing factor is chosen in such a way that information loss is minimised. Smoothing has the disadvantage of increasing the image's blurriness. This blurriness is calculated by point spread function and should be kept under acceptable limit.

1.2. SHARPENING

Sharpening is the exact opposite of smoothing where the blurriness of the image is eliminated. Visual sharpness of the image is increased by improving the focus on features, blurry regions. Local details such as edges are enhanced and therefore sharpening is used before edge detection filtering is performed. Depth information is also improved by performing sharpening.

Unsharp masking is one of the most preferred methods of sharpening which is used to enhance contrast features and borders. This study uses spatial based techniques where a sharpening filter kernel calculates the weightage at every pixel. Image sharpening techniques in the spatial domain are based on modifications of colour value of every pixel. It is modified in order to distinguish greater probability intensity levels from their neighbouring levels. One such effective method is histogram equalisation (HE). It is based on adjusting the contrast of the input image using the histogram. We can visualise and validate out results by checking the histogram graph of image before and after sharpening.

1.3. EDGE DETECTION

Filtering operation in which the boundaries can be detected of objects and person within an image. This is used widely for data extraction. This process detects any abrupt change in brightness within the image and calculates a derivative of it. These derivatives can be found by applying convolution masks in both horizontal and vertical direction. These derivatives conveys information about the gradients and orientation of the edges in the image.

Famous algorithm implemented in C++ currently are,

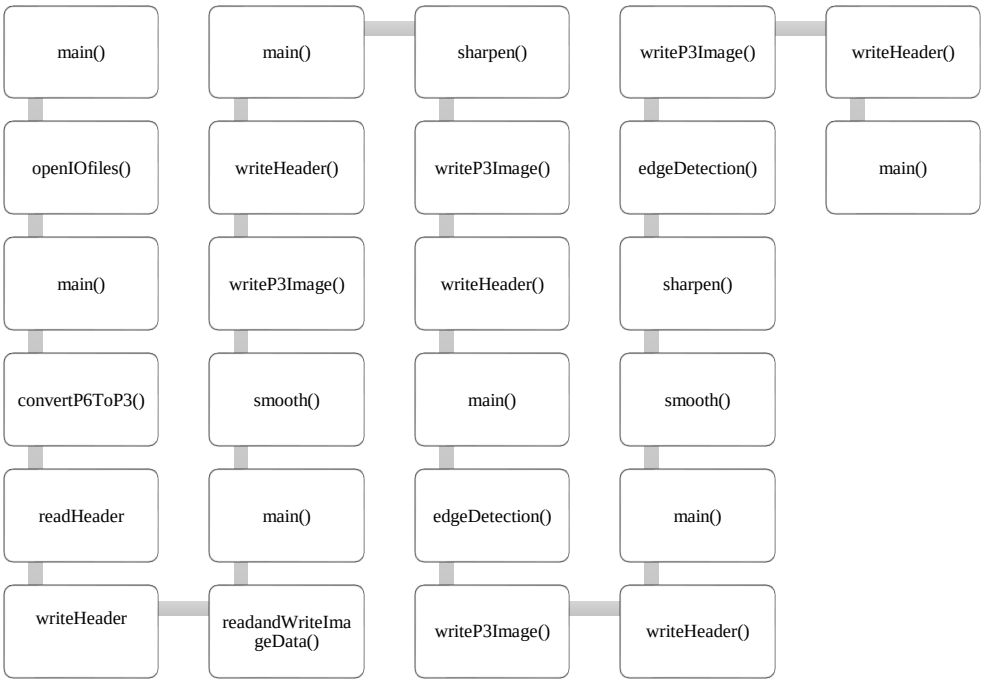
- Canny Edge detector
- Sobel Edge detector
- Prewitt Edge detector
- Roberts Cross
- Scharr Operator

1.4. WHY C++

These numerical approaches are implemented in C++. Because it does not require interpretation during runtime, C++ is a low-level language with a fast execution speed. Given the abundance of numerical libraries available in C++, makes it easier. It's also worth noting that C++ is an object-oriented programming language, which makes it easier to comprehend, maintain, and scale-up/reuse.

2. METHODOLOGY

The P6 format binary ppm image filename is obtained as the input and it is converted to P3 ASCII format which is then passed through three different filters. The control chart of the program is given below.



The functions of the UDF's described above are given in detail below.

2.1. CONVERSION OF P6 TO P3 PMM IMAGE

Input and output file streams are defined in the main function and filename is obtained from the user. The inputted filename along with both the file streams are passed through openIOfiles() function.

Void openIOfiles(ifstream& fin; ofstream& fout, char inputFilename[])

This function checks if the file with the name inputted exists and opens it. An error message is displayed if it can't find the file. The function creates another output file with the same name as input file and replaces the ".ppm" to "P3.ppm". Another error message is displayed if output file cannot be opened.

A two-dimensional vector of pixel object as an element called Image is created in the main function and convertP6ToP3() function is called.

void convertP6ToP3(ifstream& bin, ofstream& out, vector<vector<Pixel> >& image, int info[1])

The 2 file streams, Image and an integer array is passed as arguments. readHeader() is called immediately and the file streams and array are passed as arguments.

void readHeader(ifstream& fin, ofstream& fout, int imageInfo[])

The function checks for the correct format of the P6 file and also ensures that the read data is within the specified limits. The formats checked for are if the file begins with "P6", length of the comment line of the image, the maximum width of the file and max value of the pixels. Error message is displayed when one of these conditions are not proper. If no error is encountered, then the function displays the width and height of the image and also the maximum colour value of the pixel. The magic number is changed from "P6" to "P3" and writeHeader() is called.

void writeHeader(ofstream& fout, char magicNumber[], char comment[], int w, int h, int maxPixelVal)

The input arguments for this function are output stream, character array named comment, width and height of the image being converted and the maximum colour value of the pixel. The function outputs all the arguments onto the output file.

The control goes back to convertP6ToP3() function and readAndWriteImageData() is called.

void readAndWriteImageData(ifstream& fin, ofstream& fout, vector<vector<Pixel> >& image, int w, int h)

The 2 file streams, Image vector and width and height of the image is sent as arguments. The image vector is resized as per the size requirement of the image and a for-loop is used for reading the values from the binary file. An unsigned integer array called triple is created to read the pixel values and in the body of the for-loop the read value is converted firstly to unsigned char and then to unsigned int

which is then assigned to triple[]. The setPixel() function is called to transfer these values to the pixel object in the Image vector and also written to the file attached to the output stream. The loop runs until all the values are read form the binary file and is transferred to the output file and Image vector.

2.2. FILTERING OPERATONS

Different files names are created and file streams initialised to display all filter results of the image provided. Four output files are obtained after main() has been run. All the output files are in P3.ppm format and gives smooth-only, sharpen-only, edge detection-only and all the filters combined as the result. Below steps give the detailed description of the operations performed by each individual filters.

SMOOTHING

The filter used for filtering is a convolution matrix which gives the average for the top, bottom, preceding and next element.

The main function calls the smooth() function to perform this operation.

void smooth(vector<vector<Pixel> >& image)

This function takes Image vector as its only argument which already has the colour values of all the pixels of the image. A copy of image called “mat” is created so that the operations we perform does not interfere with the next set of operation needed to be done. Overloaded Assignment operator function of the Pixel class can be used to obtain the copy.

After creating a copy of the vector, the smoothing operation is done where again a for-loop is used to access every data of the vector and operator function of the Pixel class is called to manipulate the data. The values to the convolution operation is fed from the mat vector and the final values obtained is assigned to the Image vector.

SHARPENING

The matrix which perform the sharpening of the image is given below. The function is called from the main part of the program.

void sharpen(vector<vector<Pixel> >& image)

Image vector is the only argument. Like the smooth function, here also we need to create a copy “mat” for accessing the data and using the real vector for only assigning the final manipulated value. A 2-D kernel matrix used for sharpening is defined after this.

The operation involves weighted average of all the neighbouring values where the neighbours have multiplicative factor of ‘-1’ and the element to be adjusted has a factor of ‘12’. This is done for all the elements using for-loop. A 4-layered for loop is used to perform convolution of the matrix defined previously. The calculations are done with pixel values of the image obtained using getRed(), getGreen() and getBlue() function. When the inside two for loops complete the computation of the

values, the values are fed back to the main image vector using setPixel(). A lower value check is performed to ensure none of the negative integer value is fed into unsigned conversion. The outside 2 for loops makes sure that this process is done for all the elements.

EDGE DETECTION

Final image processing to be performed is edge detection. edgeDetection() from the main() is used to call this.

void edgeDetection(vector<vector<Pixel> >& image)

Image vector is the only argument and a copy of it called as “mat” is created. This discussion uses two matrices-vertical and horizontal, to perform edge detection. The derivative of the value obtained from both the matrices will be the final value. This derivative is calculated as the square root of the sum of the horizontal and vertical convolutional result.

For-loop is again used to perform this operation. The values from matrices are converted into integers before using and the 2 inside for loops calculate the horizontal and vertical convolutional values of each colour value obtained using getRed(), getGreen() and getBlue() functions. The derivative of these values are calculated outside this loop and lower value check is done. These values are then converted into unsigned and assigned to the image vector using setPixel().

2.3. DISPLAYING RESULTS

Another file is linked with the output stream which is used for displaying the results after filtering operations. This file is opened and the writeP3Image() function is called to write the Image vector to the output file.

void writeP3Image(ofstream& out, vector<vector<Pixel> >& image, char comment[], int maxColor)

The output stream of the file which has memory allocated for the output display file, Image vector with the filtered image data, comment for the P3 format and the maximum colour value of the pixel are passed to the function as arguments.

Character array is created with “P3” as the value and two int variables which had the image’s height and width stored in it. writeHeader() is called by passing the output stream, comment to be displayed, the character array, two integer variables with height and width information and the maximum value of the colour as its parameters. This function will write the basic information of the image to the file as per the basic P3 format and then exit.

A double nested for-loop is created with height and width as the outer and inner limits respectively and the Image vector is written onto the output file. If the number of elements written exceeds 10, then it gets printed in the new line.

3. VALIDATION

A small 10*10-pixel image is used in order to validate the value obtained after filtering. Pixel values from P3 output files is compared against the mathematical value computed. The main() of the code is altered to run the filtering operations separately for validation.

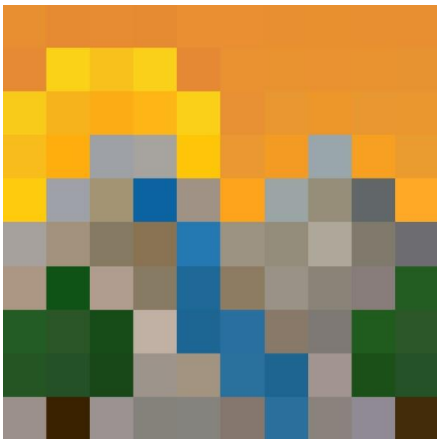


Figure 1 : Validation Image

The element (2,2)’s colour value of red pixel is compared for validating the correctness of the filtering operation. The pixel values of the original file is obtained from the “smallP3.ppm” image file and used for mathematical verification.

3.1. SMOOTHING

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

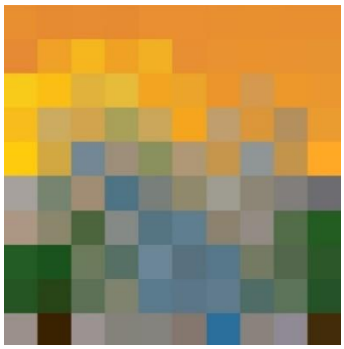


Figure 2 : Image after smoothing

The file “small_smooth.ppm” has the information of image after smooth filtering. The values for mathematical average can be obtained from “smallP3.ppm”. The closest round number is given as the result when a decimal value is obtained. K is the kernel used for smoothening operation.

For red,

= (230+231+248+247)/4 = 239

For green,

=(139+139+179+193)/4= 162.5 =162

For blue,

=(51+51+28+27)/4= 39.25= 39

Colour layer	Mathematical Value	Actual value
Red	239	239
Green	162	162
Blue	39	39

Table 1 : Validation for Smoothing

3.2. SHARPENING

$$K = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 12 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

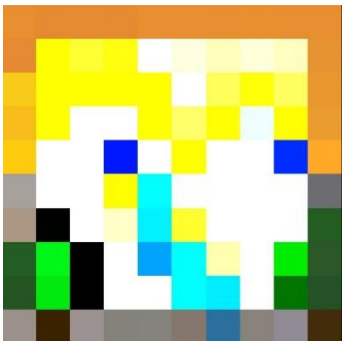


Figure 3 : Sharpened Image for validation

The file “small_sharpen.ppm” has the information of image after just sharpening and is used to obtain actual values. Here K is the kernel convolutional matrix used for sharpening.

For red,

= (12*250-(232+230+248+231+247+231+248+252)) =1081=255

For green,

=(12*211-(143+139+203+179+139+140+193+172))= 1224 =255

For blue,

=(25*12-(50+51+27+28+51+51+27+21)= -6= 0

Colour layer	Mathematical Value	Actual value
Red	255	255
Green	255	255
Blue	0	0

Table 2 : Validation for Sharpening

The value of the colour pixel are between the range 255 to 0 and are set as the upper and lower limits respectively. When one of the values exceeds beyond this limit, they are corrected.

3.3. EDGE DETECTION

The information is obtained from the “small_edgedetection.ppm” and is used for validation against values obtained from “smallP3.ppm”.

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

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

Figure 4 : Small image after Edge Detection

Using V as the vertical matrix and H as horizontal matrix and taking derivative with them,

For red,

=sqrt((232+2*231+231-248-2*247-252))^2+(232+2*230+248-231-2*248-252)^2)

=sqrt(4900+1521)=80.13=80

For green,

=sqrt((143+2*139+14-203-2*179-172)^2+(143+2*139+203-140-2*193-172)^2

=187

For red,

=sqrt((50+2*51+51-27-2*28-21)^2+(50+2*51+27-51-2*27-21)^2)

=112

Colour layer	Mathematical Value	Actual value
Red	80	79
Green	187	187
Blue	112	112

Figure 5 : Validation values for Edge Detection

4. RESULTS AND DISCUSSION

The below displayed figures are the result of respective filters applied on the stock image. The smooth and sharpen have opposite effects on the image and they are necessary in order to extract more features from the image. Smoothing causes blur and may delete few intricate features from the image whereas while sharpening, the noise may get focussed along with features that will provide us with wrong information. (Cristina Perez-Benito, 2017).

Figure 6 : Clown-Smooth

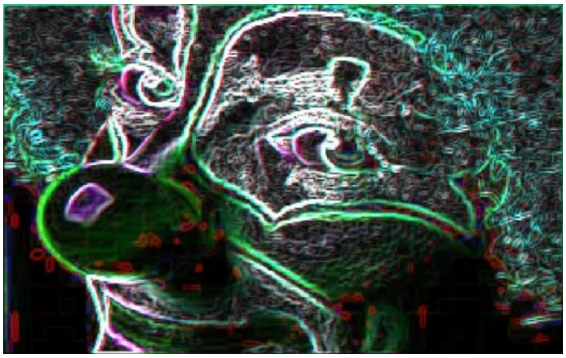


Figure 8 : Clown-Edge detection

Figure 7 : Clown-Sharpen

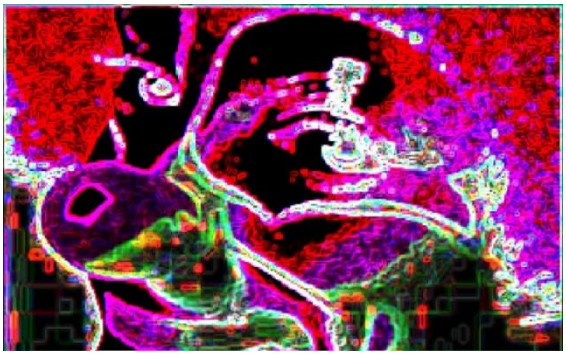


Figure 9 : Clown-All filters

Figure 10 : Dental Radiograph-Smooth

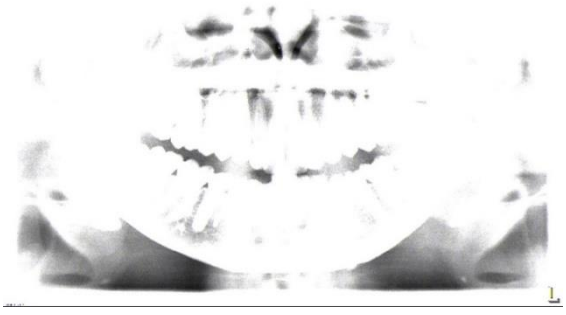


Figure 11 : Dental Radiograph-Sharpen

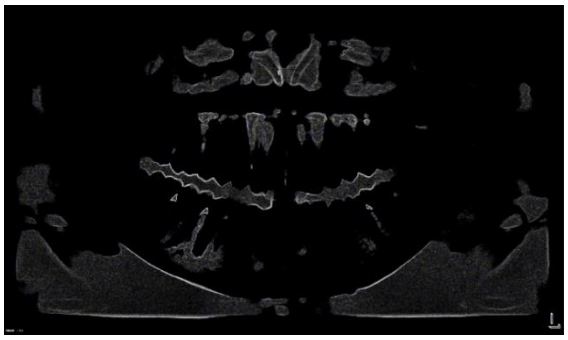


Figure 12 : Dental Radiograph-All Filters

Figure 13 : Dental Radiograph-Edge Detection

Figure 14 : m51-Smooth

Figure 16 : m51-Edge Detection

Figure 15 : m51-Sharpen

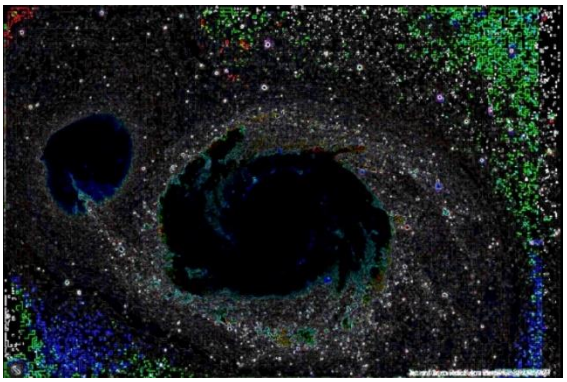


Figure 17 : m51-All Filters

Figure 18 : Photographer-Smooth

Figure 20 : Photographer-Edge Detection

Figure 19 : Photographer-Sharpen

Figure 21 : Photographer-All Filters

In the case of clown (figure : 6 to figure : 9) and dental radiography (figure : 10 to figure : 13) image sharpening operation before the edge detection has over exposed the features and several of the important information is lost. This also happens when smoothing is done where the features are under exposed. Due to the lack of noise in these two images, edge detection filter alone was able to give a better result.

The image of galaxy m51 shows mixed results when edge detection is performed with and without pre-processing. The eye of the galaxy is neatly outlined when only edge detection is performed by the features away from it gets attenuated due to noise. This is greatly tackled by pre-processing, but the eye of the galaxy is unclear here since it got over focussed as it can been clearly seen in figure : 15.

Photographer image is the best example of how to use pre-processing in order to extract better features. In comparing figure : 20 and figure : 21, we can see that even though figure : 20 gives us better information about the background details, the main focus of the image- the Photographer has lost some features. Through the application of smoothing and sharpening the figure : 21 was able to provide better details about the image such as number of buttons in the coat of photographer etc. Depending on the information required out of the image, user can choose to or choose not to implement pre-processing steps.

A perfectly sharpened image will have its contrast augmented throughout its image and this can be visualised by analysing the histogram pictures of images before and after sharpening.

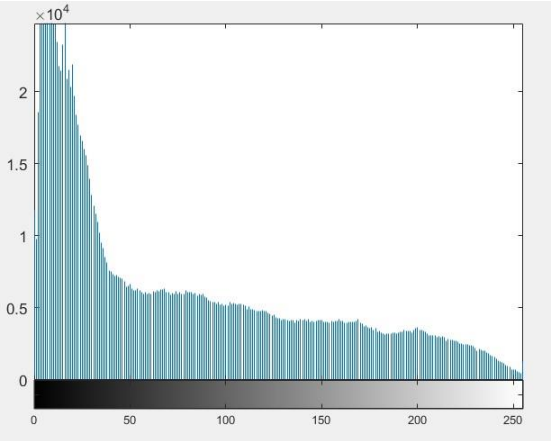


Figure 22 : Clown-Before Sharpen

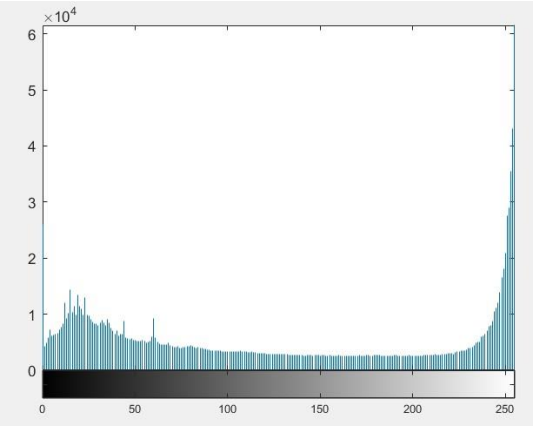


Figure 23 : Clown-After Sharpen

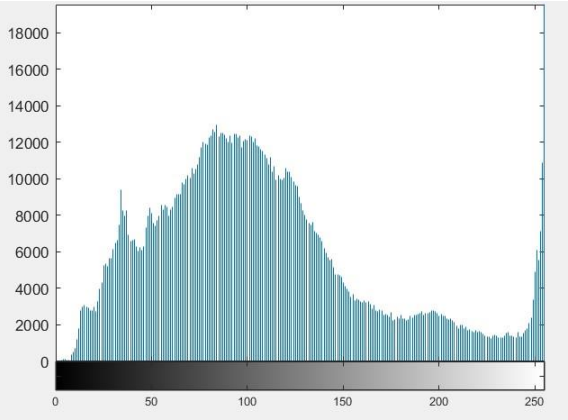


Figure 24 : Dental Radiograph-Before Sharpen

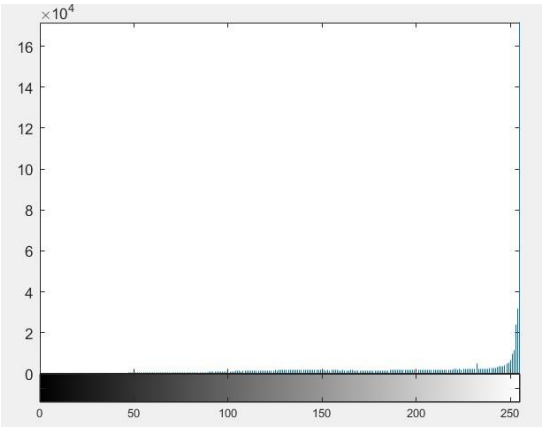


Figure 25 : Dental Radiograph-After Sharpen

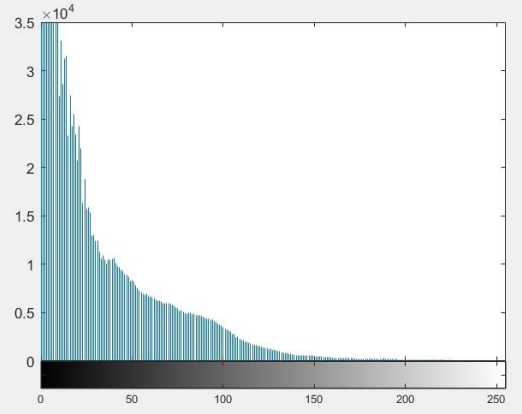


Figure 26 : m51-Before Sharpen

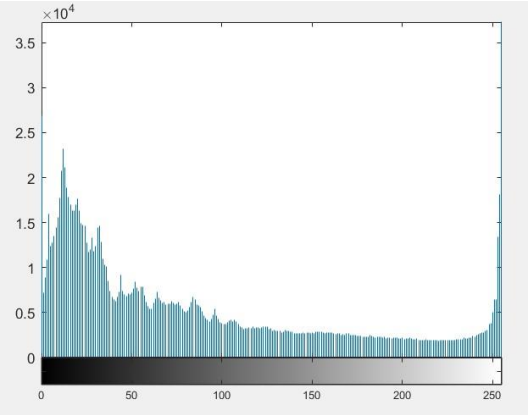


Figure 27 : m51-After Sharpen

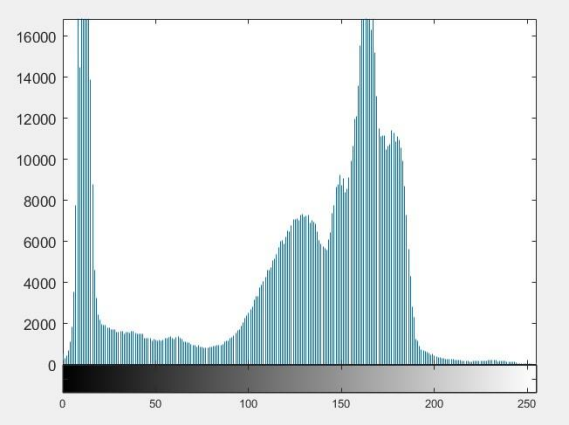


Figure 28 : Photographer-Before Sharpen

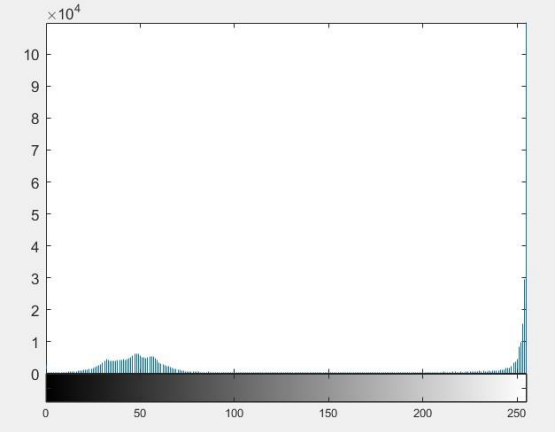


Figure 29 : Photographer-After Sharpen

The above images show that the pixel intensity distribution for sharpened images are more even than the raw images.

5. CONCLUSION

The importance and the need to perform pre-processing was discussed. A simple linear operator kernel method for smoothing, sharpening and edge detection was defined and addressed. This study was able to successfully implement and validate the following steps in C++.

- Get a P6 format ppm image and convert it into P3 ppm image
- Perform pre-processing filtering operations such as smoothing, sharpening and edge detection
- Validate the results of filtering operation using manual RGB pixel values
- Discuss the need and the order of smoothing and sharpening and elucidate their importance in image processing.

6. REFERENCES

(https://convertio.co/jpeg-ppm/)http://openseeit.sourceforge.net/. (n.d.).

Cristina Perez-Benito, S. M. (2017). Smoothing vs Sharpening of colour images: Together or Seperated. *Sciendo*.

Forsyth, P. (2001). *Computer Vision-A modern Approach.* Pearson.

GJ Awcock, R. T. (1995). *Applied Image Processing.* Macmillan.

http://netpbm.sourceforge.net/doc/ppm.html. (n.d.).

https://convertio.co/jpeg-ppm/. (n.d.).

https://uk.mathworks.com/help/images/create-image-histogram.html. (n.d.).

Maria Petrou, C. P. (2010). *Image Processing :The Fundamentals.* Wiley.

7. APPENDICES

7.1. APPENDIX-1

The stock images which are used for filtering operations in this study are given below.

Figure 30 : P3 image of Clown

Figure 31 : P3 image of Dental Radiograph

Figure 32 : P3 image of m51

Figure 33 : P3 image of Photographer

7.2. APPENDIX-2

The main() part of the function is displayed below.

```
//Include all the necessary header files required for the Program

#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <string>
#include <stdlib.h>
#include"Pixel.h"

using namespace std;

//List all the prototypes of the function to be accessed from main()

void openIOfiles(ifstream& fin, ofstream& fout, char inputFilename[]);
void readHeader(ifstream& fin, ofstream& fout, int imageInfo[]);
void convertP6ToP3(ifstream& bin, ofstream& out, vector<vector<Pixel> >& image, int info[1]);
void writeHeader(ofstream& fout, char magicNumber[], char comment[], int w, int h, int maxPixelVal);
void readAndWriteImageData(ifstream& fin, ofstream& fout, vector<vector<Pixel> >& image, int w, int h);
void writeP3Image(ofstream& out, vector<vector<Pixel> >& image, char comment[], int maxColor);
void smooth(vector<vector<Pixel> >& image);
void sharpen(vector<vector<Pixel> >& image);
void edgeDetection(vector<vector<Pixel> >& image);


/*-------------------------------------------------------------------------------------------------------------*/
/*Function name:   main                                                                                        */
/*Input Arguments : None                                                                                       */
/*Return values : 0                                                                                            */
/*-------------------------------------DESCRIPTION-----------------------------------------------------------*/
/*This function gets the value of the P6 image file name and firstly converts it into P3 format               */
/*It then calls user-defined functions to perform filtering operations on the image                            */
/*-------------------------------------------------------------------------------------------------------------*/

int main()
{
        //Initialise input and output streams for files
        ifstream f;
        ofstream o,os,osh,oe,oa;

        //Initialise the variables for the program
        char filename[100],fSmooth[100],fSharpen[100],fEdge[100],fAll[100];
        char comments[100] = "#The PPM file after smooth filtering";
        char commentsh[100] = "#The PPM file after sharpen filtering";
        char commente[100] = "#The PPM file after Edge detection filtering";
        char commentall[100] = "#The PPM file after all filtering";
        int maxcolour = 255;
        int info[3];

        //Get the name of the file to open from the user
        cout << "*"<<endl;
        cout << "FILTERING OPERATIONS IN A P6 IMAGE"<<endl;
        cout << "*"<<endl;
        cout << "Enter the name of the P6 image file: ";
        cin >> filename;
```

```
        //Pass the input and output file streams and filename for opening the file
        openIOfiles(f, o, filename);
        cout << endl << "File opened" << endl;

        //Input the name of all the output files with the input filename followed by filtering operation

        //Smoothening output file
        strcpy(fSmooth, filename);
        char* loc1 = strchr(fSmooth, '.');
        *loc1 = '\0';
        strcat(fSmooth, "smooth.ppm");

        //Sharpening output file
        strcpy(fSharpen, filename);
        char* loc2 = strchr(fSharpen, '.');
        *loc2 = '\0';
        strcat(fSharpen, "sharpen.ppm");

        //Edge Detection output file
        strcpy(fEdge, filename);
        char* loc3 = strchr(fEdge, '.');
        *loc3 = '\0';
        strcat(fEdge, "edge.ppm");

        //All filters applied
        strcpy(fAll, filename);
        char* loc = strchr(fAll, '.');
        *loc = '\0';
        strcat(fAll, "all.ppm");

        //Create a 2-D vector with Pixel object as an element
        vector<vector<Pixel>> Image;

        //Call this function to convert P6 to P3 image
        convertP6ToP3(f, o, Image, info);

        //Close the unnecessary files to free up memory
        f.close();

        //Duplicate the image vector to display the results of filtering operations separately
        vector<vector<Pixel>> Smooth;
        vector<vector<Pixel>> Sharpen;
        vector<vector<Pixel>> Edge;
        Smooth = Image;
        Sharpen = Image;
        Edge = Image;

        //Open the output files in which filtering results have to displayed and call the corresponding function
followed by writing function

        //Smoothing
        os.open(fSmooth);
        smooth(Smooth);
        writeP3Image(os, Smooth, comments, maxcolour);
        os.close();

        //Sharpening
        osh.open(fSharpen);
        sharpen(Sharpen);
```

```
        writeP3Image(osh, Sharpen, commentsh, maxcolour);
        osh.close();

        //Edge Detection
        oe.open(fEdge);
       edgeDetection(Edge);
        writeP3Image(oe, Edge, commente, maxcolour);
        oe.close();

        //All Filtering Operations
        oa.open(fAll);
        smooth(Image);
        sharpen(Image);
        edgeDetection(Image);
        writeP3Image(oa, Image, commentall, maxcolour);
        oa.close();

        //Returning 0
        return 0;
}
```

7.3. APPENDIX-3

The below codes are the Pixel class header and its function definition

CODE-1

```
/*-------------------------------------------------------------------*/
/* Pixel class declaration.                                          */
/* File name: pixel.h                                                */
/* This class implements the concept of a pixel                      */
//#ifndef PIXEL_H
//#define PIXEL_H
#include <iostream> //Required for istream, ostream
using namespace std;

class Pixel
{
  public:
        // constants to enable undeflow/overflow detection in rgb values
        static const unsigned int MAXVAL = 255;
        static const unsigned short RMASK = 4;
        static const unsigned short GMASK = 2;
        static const unsigned short BMASK = 1;
        static const unsigned short CHECK = 7;

        //Constructors
        Pixel(); //Default
        Pixel(unsigned ); //Grey scale
        Pixel(unsigned,unsigned,unsigned); //Full color range

        // Accessor Methods
        unsigned getRed() const {return red;}
        unsigned getGreen() const {return green;}
        unsigned getBlue() const {return blue;}

        // Mutator Methods
        Pixel& setPixel(unsigned r, unsigned g, unsigned b);
        Pixel& setRed(unsigned r);
        Pixel& setGreen(unsigned g);
        Pixel& setBlue(unsigned b);

        //Overloaded operators.
        //Addition.
        Pixel operator+(const Pixel& p) const;

        //Multiplication of a Pixel by a floating point value.
        Pixel operator*(double v) const;

        //Division of a Pixel by an integer value.
        Pixel operator/(unsigned v) const;

        //Input operator.
        friend istream& operator >>(istream& in, Pixel& p);

        //Output operator.
        friend ostream& operator <<(ostream& out, const Pixel& p);

        // assignment
        Pixel& operator=(const Pixel& rhs);
```

```
        bool overflow(); //check for overflow
        void reset();   //reset to MAXVAL

  private:
        void validate(); //set overflow bits
        unsigned int red, green, blue;
        unsigned short overflowFlag;
};

//#endif
/*-------------------------------------------------------------------*/
```

CODE-2

```
#include "Pixel.h"
/*---------------------------------------------------------------*/
/* Pixel class implementation.                                   */
/* File name: Pixel.cpp                                          */

/* Addition (+) operator.                                        */
Pixel Pixel::operator+(const Pixel& p) const
{
  Pixel temp;
  temp.red = red + p.red;
  temp.green = green + p.green;
  temp.blue = blue + p.blue;
  temp.validate();
  return temp;
}
/*---------------------------------------------------------------*/
/* Multiplication (*) operator.                                  */
Pixel Pixel::operator*(double v) const
{
  Pixel temp;
  temp.red = static_cast<unsigned int>(red*v);
  temp.green = static_cast<unsigned int>(green*v);
  temp.blue = static_cast<unsigned int>(blue*v);
  temp.validate();
  return temp;
}
/*---------------------------------------------------------------*/
/* Division (/) operator.                                        */
Pixel Pixel::operator/(unsigned int v) const
{
  Pixel temp;
  temp.red = red/v;
  temp.green = green/v;
  temp.blue = blue/v;
  temp.validate();
  return temp;
}
/*---------------------------------------------------------------*/
/* Output << operator.                                           */
ostream& operator<<(ostream& out, const Pixel& p)
{
  out << p.red << ' ';
  out << p.green << ' ';
  out << p.blue;
  return out;
}
/*---------------------------------------------------------------*/
/* Input (>>) operator.                                          */
istream& operator>>(istream& in, Pixel& p)
{
  in >> p.red >> p.green >> p.blue;
  p.validate();
  return in;
}
/*---------------------------------------------------------------*/
/* default constructor, sets the rgb to 0.                       */
Pixel::Pixel()
{
```

```
  //Black
  red=green=blue=0;
  return;
}
/*---------------------------------------------------------------*/
/* alternate constructor 1, sets the rgb to value.               */
Pixel::Pixel(unsigned int value)
{
  //Grey scale
  red=green=blue=value;
  validate();
  return;
}
/*---------------------------------------------------------------*/
/* alternate constructor 2, sets the rgb value to r,g,b.         */
Pixel::Pixel(unsigned int r,unsigned int g,unsigned int b)
{
  //Full color range
  red=r;
  green=g;
  blue=b;
  validate();
  return;
}
/*---------------------------------------------------------------*/
/*  overloaded assignment operator                               */
Pixel& Pixel::operator=(const Pixel& rhs)
{
  if(this != &rhs)
  {
    red = rhs.red;
    green = rhs.green;
    blue = rhs.blue;
    overflowFlag = rhs.overflowFlag;
    validate();
    return *this;
  }
  else return *this;
}


/*---------------------------------------------------------------*/
/* set the rgb value of a pixel                                  */
Pixel& Pixel::setPixel(unsigned r, unsigned g, unsigned b)
{
        red = r;
        green = g;
        blue = b;
        return *this;
}


/*---------------------------------------------------------------*/
/* set the red value of a pixel                                  */
Pixel& Pixel::setRed(unsigned r)
{
        red = r;
        return *this;
}
```

```
/*-----------------------------------------------------------------*/
/* set the green value of a pixel                                  */
Pixel& Pixel::setGreen(unsigned g)
{
        green = g;
        return *this;
}


/*-----------------------------------------------------------------*/
/* set the blue value of a pixel                                   */
Pixel& Pixel::setBlue(unsigned b)
{
        blue = b;
        return *this;
}


/*-----------------------------------------------------------------*/
/* check for overflow                                              */
bool Pixel::overflow()
{
 return(overflowFlag&CHECK);
}

/*-----------------------------------------------------------------*/
/* reset the rgb values to 255 (white)                             */
void Pixel::reset()
{
 if(red > MAXVAL) red = MAXVAL;
 if(green > MAXVAL) green = MAXVAL;
 if(blue > MAXVAL) blue = MAXVAL;
 overflowFlag = 0;
}

/*-----------------------------------------------------------------*/
/* set the overflow flag                                           */
void Pixel::validate()
{
 if(red > MAXVAL) overflowFlag = overflowFlag|RMASK;
 if(green > MAXVAL) overflowFlag = overflowFlag|GMASK;
 if(blue > MAXVAL) overflowFlag = overflowFlag|BMASK;
}
```

7.4. APPENDIX-4

The conversion codes used to convert P6.ppm image binary format to P3.ppm ascii format. It is also used to write image details into a file in P3 format.

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <string>
#include <stdlib.h>
#include"Pixel.h"

void openIOfiles(ifstream& fin, ofstream& fout, char inputFilename[]);
void readHeader(ifstream& fin, ofstream& fout, int imageInfo[]);
void convertP6ToP3(ifstream& bin, ofstream& out, vector<vector<Pixel> >& image, int info[1]);
void writeHeader(ofstream& fout, char magicNumber[], char comment[], int w, int h, int maxPixelVal);
void readAndWriteImageData(ifstream& fin, ofstream& fout, vector<vector<Pixel> >& image, int w, int h);
void writeP3Image(ofstream& out, vector<vector<Pixel> >& image, char comment[], int maxColor);

const int MAXWIDTH(5);
const int MAXLEN(1000);
const char newline('\n');
const char terminator('@');
const char nullChar('\0');

/*------------------------------------------------------------------------------------------------------------ */
/*Function name:   openIOfiles                                                                                 */
/*Input Arguments : Input and Output stream of a file, filename                                                */
/*Return values : none                                                                                         */
/*---------------------------------------DESCRIPTION-----------------------------------------------------*/
/* The functions gets the name of the file to be opened and checks if it can be opened. Error  */
/*message is displayed if it can't be opened. Name of the output file is derived from input file */
/*and opened. A check is done if it is opened properly and error is displayed if not.          */
/*-----------------------------------------------------------------------------------------------------------*/

void openIOfiles(ifstream& fin, ofstream& fout, char inputFilename[])
{
        //Input file is opened
        fin.open(inputFilename, ios::binary);

        //Output file is initialised
        char* outputFilename;
        outputFilename = (char*)malloc(sizeof(inputFilename) + 10);

        //Check for proper opening of input file
        if (!fin)
        {
                cout << "could not open file: bye" << endl;
                exit(1);
        }

        //Output file is named and opened from input file plus the output format of the image
        strcpy(outputFilename, inputFilename);
        char* loc = strchr(outputFilename, '.');
        *loc = '\0';
        strcat(outputFilename, "P3.ppm");
        fout.open(outputFilename);
```

```
	//Check for proper opening of output file
	if (!fout)
	{
		cout << "cannot open output file: bye" << endl;
	}
}

/*-------------------------------------------------------------------------------------------------------------*/
/*Function name:   convertP6ToP3                                                                               */
/*Input Arguments : Input and output stream, 2-D vector, int array                                             */
/*Return values : none                                                                                         */
/*---------------------------------------DESCRIPTION-----------------------------------------------------------*/
/* The function is used to convert P6 file to P3 format and calls various UDF for the purpose  */
/*-------------------------------------------------------------------------------------------------------------*/

void convertP6ToP3(ifstream& bin, ofstream& out, vector<vector<Pixel> >& image, int info[1])
{
	//Calling the function to read the header of the file
	readHeader(bin, out, info);

	int width = info[0], height = info[1];

	//this function reads and prints the data to the vector and output file
	readAndWriteImageData(bin, out, image, width, height);
}

/*-------------------------------------------------------------------------------------------------------------*/
/*Function name:   readHeader                                                                                  */
/*Input Arguments : Input and output stream, int array                                                         */
/*Return values : none                                                                                         */
/*-----------------------------------DESCRIPTION---------------------------------------------------------------*/
/* The function is used to read the header from the P6 file and checks for the format. It exits  */
/* the program if improper format is detected. write header is called to write the info in the    */
/* output file                                                                                                  */
/*-------------------------------------------------------------------------------------------------------------*/

void readHeader(ifstream& fin, ofstream& fout, int imageInfo[])
{
	// define and initialise input variables
	char bData[MAXLEN] = { 0 }, magicNumber[MAXWIDTH], comment[MAXLEN] = { "#" };
	int bIndex = 0, charCount = 0, infoCount = 0;
	char ch, aNumber[MAXWIDTH];

	// input first line of text header(magic number)
	// if the magic number is not P6 exit the program
	fin.getline(magicNumber, 3);
	if (strcmp(magicNumber, "P6") != 0)
	{
		cout << "unexpected file format\n";
		exit(1);
	}

	// clear bData array and reset bIndex
	// input next line of text header
	strcpy(bData, " ");
	bIndex = 0;

	fin.getline(bData, MAXLEN);

	do {
```

```
		// is this the beginning of a comment
		ch = bData[bIndex];
		if (ch == '#')
		{
			// comment has been read
			// get all characters until a newline is found
			charCount = 0;
			while (ch != terminator && charCount < MAXLEN)
			{
				comment[charCount] = ch;
				++bIndex;
				++charCount;
				ch = bData[bIndex];
				cout << ch;
			}

			if ((charCount == MAXLEN) && (ch != newline))
			{
				cout << "Comment exceeded max length of " << MAXLEN << endl;
				exit(1);
			}

			// get the next line of data
			strcpy(bData, " ");
			bIndex = 0;
			fin.seekg(1, fin.cur);
			fin.getline(bData, MAXLEN);

		}
		else
		{
			// this is not a comment
			// parse bData for image information
			charCount = 0;
			// look past whitespace
			while (bIndex < MAXLEN && isspace(bData[bIndex]))
			{
				++bIndex;
			}
			// may be the beginning of a decimal value
			while (bIndex < MAXLEN && isdigit(bData[bIndex]))
			{
				aNumber[charCount] = bData[bIndex];
				++bIndex;
				++charCount;
				if (charCount == MAXWIDTH)
				{
					cerr << "Maximum width of " << MAXWIDTH << " digits was
exceeded.. " << endl;
					exit(1);
				}
			}

			// look at size of aNumber
			if (charCount > 0)
			{
				// we have image information, terminate string
				aNumber[charCount] = nullChar;
				// convert from ascii to integer
				imageInfo[infoCount] = atoi(aNumber);
```

```
                                        ++infoCount;
                                        // verify input
                                        switch (infoCount)
                                        {
                                        case 1: cout << "A width of " << imageInfo[infoCount - 1] << " has been
read " << endl;
                                                break;
                                        case 2: cout << "A height of " << imageInfo[infoCount - 1] << " has bene
read " << endl;
                                                break;
                                        case 3: cout << "Maxcolor of " << imageInfo[infoCount - 1] << " has been
read " << endl;
                                                break;
                                        }
                                }
                                else if (infoCount < 3)
                                {
                                        // aNumber has 0 digits and infoCount < 3
                                        // we need more image information
                                        // get next line of data and parse for image information
                                        strcpy(bData, " ");
                                        bIndex = 0;
                                        fin.getline(bData, MAXLEN);
                                        //fin.seekg(1, fin.cur);

                                }
                        }
        } while (infoCount < 3 && !fin.eof());
        if (infoCount < 3)
        {
                cerr << "image information could not be found " << endl;
                exit(1);
        }
        // we have all of the information
        // write header to ascii file
        strcpy(magicNumber, "P3");
        writeHeader(fout, magicNumber, comment, imageInfo[0], imageInfo[1], imageInfo[2]);
}

/*-------------------------------------------------------------------------------------------------------*/
/*Function name:   writeHeader                                                                           */
/*Input Arguments : Output stream, character array with "P3" and custom comment,   */
/*                3 integers with width and height of the image and max pixel value          */
/*Return values : none                                                                                   */
/*---------------------------------------DESCRIPTION-----------------------------------------*/
/* The function displays the input parameters received into output file                    */
/*-------------------------------------------------------------------------------------------------------*/

void writeHeader(ofstream& fout, char magicNumber[], char comment[], int w, int h, int maxPixelVal)
{
        // write image imformation to output file
        fout << magicNumber << newline;
        fout << comment << newline;
        fout << w << ' ' << h << ' ' << maxPixelVal << newline;
}

/*-------------------------------------------------------------------------------------------------------------*/
/*Function name:   readAndWriteImageData                                                                       */
/*Input Arguments : Input and output stream, 2-D vector of image data, image height and width   */
/*Return values : none                                                                                         */
```

```
/*-------------------------------------DESCRIPTION-----------------------------------------------------------*/
/* The function reads the value from binary P6 file and assigns it to image vector using                    */
/* setPixel()and output file.                                                                               */
/*-----------------------------------------------------------------------------------------------------------*/

void readAndWriteImageData(ifstream& fin, ofstream& fout, vector<vector<Pixel> >& image, int w, int h)
{
	// read and write image data
	// define input variables

	int charCount = 0;
	char colorByte;
	unsigned char aChar;
	unsigned int triple[3];   // red, green, blue

	// allocate memory
	image.resize(h); // allocate h rows
	fin.seekg(0, fin.cur);

	for (int i = 0; i < h; i++)
	{
		image[i].resize(w);// for each row allocate w columns

		for (int j = 0; j < w; j++)
		{
			for (int k = 0; k < 3; k++)
			{
				// read one byte
				fin.read(&colorByte, 1);

				// convert to unsigned char
				aChar = (unsigned char)colorByte;

				// save as unsigned int
				triple[k] = (unsigned int)aChar;

				// write as int
				fout << triple[k] << ' ';
			}
			// CR printed over 10 pixels
			++charCount;
			if (charCount == 10)
			{
				fout << "\r\n";
				charCount = 0;
			}
			image[i][j].setPixel(triple[0], triple[1], triple[2]);
		}
	}
}

/*-----------------------------------------------------------------------------------------------------------*/
/*Function name:   writeP3Image                                                                              */
/*Input Arguments : Output stream, 2-D vector of image data, char comment to be displayed in                 */
/*                the image and max value of the pixel                                                     */
/*Return values : none                                                                                       */
/*-------------------------------------DESCRIPTION-----------------------------------------------------------*/
/* The function writes the information from image vector to the output file                                  */
/*-----------------------------------------------------------------------------------------------------------*/
```

```
void writeP3Image(ofstream& out, vector<vector<Pixel> >& image, char comment[], int maxColor)
{
	//Initialise values
	int h, w, pCount(0);
	char magicNumber[3] = "P3";

	h = (int)image.size();
	w = (int)image[0].size();

	//Call the writeHeader function to display the correct details of P3 format in output file
	writeHeader(out, magicNumber, comment, w, h, maxColor);

	//Display the RGB pixel values into the file
	for (int i = 0; i < h; i++)
	{
		for (int j = 0; j < w; j++)
		{
			out << image[i][j];
			++pCount;
			if (pCount == 10)
			{
				out << "\r\n";
				pCount = 0;
			}
			else out << ' ';
		}
	}
}
```

7.5. APPENDIX-5

The codes used to apply filter to the images are given below.

```
//Include all the necessary files for the program

#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include <string>
#include <stdlib.h>
#include"Pixel.h"

//Funtion identifiers are listed below

void smooth(vector<vector<Pixel> >& image);
void sharpen(vector<vector<Pixel> >& image);
void edgeDetection(vector<vector<Pixel> >& image);

const int MAXWIDTH(5);
const int MAXLEN(1000);
const char newline('\n');
const char terminator('@');
const char nullChar('\0');

/*-------------------------------------------------------------------------------------------------------------*/
/*Function name:   smooth                                                                                      */
/*Input Arguments : 2-D vector with Pixel object                                                               */
/*Return values : none                                                                                         */
/*-----------------------------------DESCRIPTION-----------------------------------------------------------*/
/* The functions gets vector matrix as the parameter which has the information about the image  */
/*the RGB pixel values are manipulated by the given kernel function and the input vector is       */
/*modified                                                                                                     */
/*-------------------------------------------------------------------------------------------------------------*/

void smooth(vector<vector<Pixel> >& Image)
{
        //Creating a copy of the input 2-D vector
        int h = Image.size();
        int w = Image[0].size();
        double val = 4;
        vector<vector<Pixel>> mat;
        mat2 = Image;

        //Creating a Pixel object to store intermediate values
        Pixel sum;

        //A double nested for loop is created and the body contains the logic to manipulate all the elements
        for (int i = 1; i < h - 1; i++)
                for (int j = 1; j < w - 1; j++)
                {
                        //Code for calculation and assignment of the value
                        sum = (mat[i + 1][j]) + (mat[i - 1][j]) + (mat[i][j + 1]) + (mat[i][j - 1]);
                        sum = (sum / (val));
                        Image[i][j] = (sum);

                        //This Pixel function makes sure the RGB values are under the MAXVAL limit
                        Image[i][j].reset();
```

```
		}

}

/*-------------------------------------------------------------------------------------------------------------*/
/*Function name:   sharpen                                                                                     */
/*Input Arguments : 2-D vector with Pixel object                                                               */
/*Return values : none                                                                                         */
/*-------------------------------------DESCRIPTION-----------------------------------------------------------*/
/* The functions gets vector matrix as the parameter which has the information about the image  */
/*to be sharpened. The values are converted to int before getting manipulated by kernel matrix.  */
/*It is again converted back to unsigned before getting assigned to Pixel object                 */
/*-------------------------------------------------------------------------------------------------------------*/

void sharpen(vector<vector<Pixel> >& image)
{
	//Initialise the values required for the program
	int r = 0;
	int g = 0;
	int b = 0;
	unsigned int triple[3];

	int h = image.size();
	int w = image[0].size();

	//Kernel matrix used for Sharpening
	int sharpen[3][3] = { {-1,-1,-1},{-1,12,-1}, {-1,-1,-1} };

	//Creating a copy of vector matrix
	vector<vector<Pixel>> mat2;
	mat2 = image;

	//Four layered For loop is used to change the RGB pixel values according to sharpen kernel
	for (int i = 1; i < h - 1; i++)
		for (int j = 1; j < w - 1; j++)
		{
			for (int k = 0; k < 3; k++)
			{

				for (int l = 0; l < 3; l++)
				{
					//RGB values are calculated separately
					int r1 = (int)(mat2[i + k - 1][j + l - 1].getRed());
					r = r + r1 * sharpen[k][l];

					int g1 = (int)(mat2[i + k - 1][j + l - 1].getGreen());
					g = g + g1 * sharpen[k][l];

					int b1 = (int)(mat2[i + k - 1][j + l - 1].getBlue());
					b = b + b1 * sharpen[k][l];

				}

			}

			//The unsigned value of a negative number is a large positive number and will
produce errors
			//If the RGB value is negative, it gets assigned the least possible value
			if (r < 0)
				r = 0;
```

```
                        if (g < 0)
                                g = 0;

                        if (b < 0)
                                b = 0;

                        //Converting int to unsigned for assigning to Pixel object
                        triple[0] = (unsigned)r;
                        triple[1] = (unsigned)g;
                        triple[2] = (unsigned)b;

                        //setPixel() funtion assigns the passed value to RGB of Pixel Object respectively
                        image[i][j].setPixel(triple[0], triple[1], triple[2]);

                        //This Pixel funtion makes sure the RGB values are under the MAXVAL limit
                        image[i][j].reset();

                        //resetting values for next loop
                        r = g = b = 0;
                }
}

/*-------------------------------------------------------------------------------------------------------------*/
/*Function name:   edgeDetection                                                                               */
/*Input Arguments : 2-D vector with Pixel object                                                               */
/*Return values : none                                                                                         */
/*---------------------------------------DESCRIPTION-----------------------------------------------------------*/
/* The functions gets vector matrix as the parameter which has the information about the image  */
/*These values are manipulated by 2 kernels-horizontal and vertical. The derivative of both the   */
/*values are calculated and assigned as the new value                                                          */
/*-------------------------------------------------------------------------------------------------------------*/

void edgeDetection(vector<vector<Pixel> >& image)
{
        //Initialise the values required for the program
        int r = 0, rh = 0, rv = 0;
        int g = 0, gh = 0, gv = 0;
        int b = 0, bh = 0, bv = 0;

        unsigned int triple[3];

        int h = image.size();
        int w = image[0].size();

        //Horizontal and Vertical Kernel
        int hor[3][3] = { {1,0,-1},{2,0,-2},{1,0,-1} };
        int ver[3][3] = { {1,2,1},{0,0,0},{-1,-2,-1} };

        //Creating a copy of the input vector
        vector<vector<Pixel>> mat2;
        mat2 = image;

        //Using for loops to calculate the horizontal and vertical pixel values of the vector
        for (int i = 1; i < h - 1; i++)
        {
                for (int j = 1; j < w - 1; j++)
                {
                        for (int k = 0; k < 3; k++)
                        {
```

```
                        for (int l = 0; l < 3; l++)
                        {
                                int r1 = (int)(mat2[i + k - 1][j + l - 1].getRed());
                                rh = rh + r1 * hor[k][l];
                                rv = rv + r1 * ver[k][l];

                                int g1 = (int)(mat2[i + k - 1][j + l - 1].getGreen());
                                gh = gh + g1 * hor[k][l];
                                gv = gv + g1 * ver[k][l];

                                int b1 = (int)(mat2[i + k - 1][j + l - 1].getBlue());
                                bh = bh + b1 * hor[k][l];
                                bv = bv + b1 * ver[k][l];

                        }

                }

                //Calculating the derivative of both kernels
                r = sqrt(pow(rh, 2) + pow(rv, 2));
                g = sqrt(pow(gh, 2) + pow(gv, 2));
                b = sqrt(pow(bh, 2) + pow(bv, 2));

                //The unsigned value of a negative number is a large positive number and will
produce errors
                //If the RGB value is negative, it gets assigned the least possible value
                if (r < 0)
                        r = 0;

                if (g < 0)
                        g = 0;

                if (b < 0)
                        b = 0;

                //Converting int to unsigned for assigning to Pixel object
                triple[0] = (unsigned)r;
                triple[1] = (unsigned)g;
                triple[2] = (unsigned)b;

                //setPixel() funtion assigns the passed value to RGB of Pixel Object respectively
                image[i][j].setPixel(triple[0], triple[1], triple[2]);

                //This Pixel funtion makes sure the RGB values are under the MAXVAL limit
                image[i][j].reset();

                //resetting values for next loop
                r = rv = rh = g = gh = gv = b = bh = bv = 0;

        }
    }
}
```