PROCESAMIENTO DE IMÁGENES PPM Y PGM EN C++

Salazar, Ángelo., Revelo, Juan José., Bailon, Juan José.
angelo.salazar@correounivalle.edu.co
juan.jose.revelo@correounivalle.edu.co
juan.bailon@correounivalle.edu.co
Universidad del Valle.

Resumen— La aparición de C++ como lenguaje de programación marcó el camino del futuro de la programación orientada a objetos abriendo así las puertas al desarrollo de software más especializado, el procesamiento digital de imágenes es uno de los principales objetos de estudio de la actualidad el cual es un campo abierto de la investigación vinculando áreas como la computación, y las matemáticas se plantea y desarrolla un programa orientado objetos el cual permite trabajar con imágenes de formato PPM y PGM en la cual haciendo uso del lenguaje pueden ser modificadas y estudiadas.

Índice de Términos— C++, Procesamiento de imágenes, Lenguaje de programación, Programación orientada a objetos.

I. Introducción

La programación orientada a objetos tiene como precursor a C++ un lenguaje diseñado en 1979 por Bjarne Stroustrup llamado inicialmente "*C with classes"* fue diseñado inicialmente para proveer facilidades al momento de programar entre Simula y C hacerlo más flexible y eficiente luego su intención se siguió extendiendo al exitoso lenguaje de C, con mecanismos que permitieran la manipulación de objetos. Suele decirse que C++ es un lenguaje de programación multiparadigma ya que se admite la programación estructurada y la programación orientada a objetos . En la actualidad es un lenguaje versátil, potente y de fácil acceso. Los programadores profesionales lo posicionan como la herramienta número uno para el desarrollo de aplicaciones debido a la riqueza de operadores, flexibilidad, eficiencia y consistencia.

El análisis y procesamiento de imágenes generalmente se distingue como una área de la ingeniería que permite la extracción de mediciones y datos de una imagen debido a que como parámetro de entrada es una imagen y su resultado comúnmente es numérico.

Para llegar a la información y parámetros deseados es necesario pasar por procesamientos donde se analiza la imagen y se adecua para el método específico, los cuales se han hecho en respuesta a problemáticas comunes de las imágenes. El método llamado a utilizar será el de algoritmos en el dominio Espacial que se refiere a métodos que procesan una imagen pixel por pixel, o también tomando en cuenta un conjunto de pixeles vecinos.

También tendremos en cuenta los tipos de transformaciones.

- *Transformaciones Puntuales.* Las cuales el píxel resultante de la operación depende solo del valor del pixel de entrada. estas incluyen la manipulación de los píxeles uno a uno, por ejemplos la segmentación, la binarización, la corrección de color, tono saturación, gamma,etc.

- *Transformaciones locales.* para obtener el píxel de salida se utiliza los píxeles vecinos en la operación. Como ejemplo sería suavizado, media, operaciones morfológicas, realce de bordes.

- *Transformaciones geométricas.* Se toma en cuenta la posición de los pixeles en la imagen y se aplica operaciones de traslación/rotación, los ejemplos más

comunes son cambios de escala, rotación, traslación, rectificación y transformaciones de los pixeles.

El portable pixel map o PPM como comúnmente se conoce es un tipo de archivo de imagen crudo el cual se suele representar por medio de matrices y puede ser visto desde un texto plano como un txt o cualquier otro editor de texto. Las imágenes PPM están compuestas por lo siguiente:

1. El "número mágico" el cual permite como identificador del tipo de archivo. Este número es de dos caracteres y para imágenes ppm este suele ser "P3" o "P6".
2. Una altura y un ancho el cual representa el tamaño de la imagen en pixeles.
3. El máximo valor que puede tomar un píxel, comúnmente se usa 255.
4. El cuerpo de la imagen.

El cuerpo de la imagen está representado por un mapa de bits en orden de arriba hacia abajo. Cada fila está conformada por el ancho de los pixeles en orden de izquierda a derecha, cada píxel se compone de tres más siguiendo el orden rojo, verde y azul (RGB), cada archivo es representado en código ASCII.

II. Requerimientos

Se debe lograr hacer operaciones y cálculos matriciales sobre imágenes crudas de formato PPM (Portable Pixel Map) y PGM (Portable Graymap), las cuales son imágenes a color y en escala de grises respectivamente, lo que permite trabajar con ellas a partir del programa que se desarrollará, aplicando operaciones tales como:

- Realizar operaciones que cambien las características visuales de la imagen, como por ejemplo sus colores.

- Lograr combinar dos imágenes entre sí brindando unos resultados agradables a la vista, por ejemplo cambiar el fondo de una imagen.

- Hacer el cálculo que determina la disparidad binocular en imágenes estéreo, obtenidas desde los dos puntos focales situados a distancia B y que capturan imágenes desde un punto p, con distancia Z, logrando diferenciar las imágenes que aunque son similares no son iguales

- Realizar operaciones sobre las imágenes sin importar el formato, por ejemplo cambiar su tamaño, el número de píxeles y sus colores.

III. Procedimiento

Para llevar a cabo el desarrollo del proyecto se aplican los conocimientos teóricos y prácticos del curso de Programación orientada a objetos, buscando así desde un principio como primer paso identificar los atributos, los métodos y las clases necesarias para el funcionamiento del programa. Se debe comprender la relación entre las clases y los métodos así como también las relaciones entre clases y el paso de mensajes entre sí para tener coherencia al momento de programar.

Este proyecto de programación se busca llevar a cabo usando siguientes herramientas:

- ➢ Google Drive
- ➢ Visual Studio Code
- ➢ Repl.it
- ➢ GitHub como repositorio y GitHub desktop
- ➢ Discord

Estas herramientas permiten tener acceso a la información necesaria alojada en la nube (Google drive), así como también un entorno de trabajo y programación con extensiones que facilitan la visualización y agilidad al momento de programar (Visual Studio Code), se debe llevar un registro de lo cambios realizados y las versiones del proyecto para así tener presente los avances y evitar posibles errores o pérdidas de código por lo tanto se debe

hacer uso de un repositorio (GitHub), para facilitar la comunicación entre los integrantes del grupo ***Mr.Repo*** se hace uso de aplicaciones que permitan la transferencia de datos (Google Drive) y el chat de voz (Discord).

IV. Filtros y Métodos

Los filtros son una de las herramientas más utilizadas hoy en día en las imágenes ya que estos permiten tener cambios sobre los mismos dándole diferentes aspectos, hoy en día el avance de los filtros ha permitido que sean aplicados inclusive en vídeos y que la cantidad de filtros existentes superen los miles.

Para la ejemplificación y poder observar el resultado de la aplicación de los filtros se usará como ejemplo la imagen de Lenna.

Imagen 1. *Imagen de Lenna, usada comúnmente en el procesamiento de imágenes.*

a) Sepia:

El filtro sepia es una de las herramientas más relevantes en la edición de imágenes. El sepia también es conocida como la alta pigmentación en tonos cafés, el filtro sepia brinda a las imágenes un tono cálido.

Para aplicar el filtro a partir de una imagen RGB en formato PPM cada pixel debe ser multiplicado por unos valores específicos o una matriz conocida como kernel.

[0.393 0.769 0.189]
[0.349 0.686 0.168]
[0.272 0.534 0.131]

Cada componente de la imagen RGB es multiplicada por el kernel, y después suma sus componentes para obtener un pixel, este después es reemplazado en el vector principal de la imagen para tener la salida en formato PPM. Para esto se debe seguir el siguiente algoritmo. ***Ver anexo 1.***

Imagen 2. *Filtro sepia aplicado en imagen de Lenna.*

b) Gris:

El gris es un filtro de tono de grises que permite convertir una imagen PPM a escala de grises brindando así a la imagen un aspecto de antigüedad y elegancia, este filtro es muy conocido y relevante en el mundo de la fotografía, inclusive las primeras fotos tomadas en la historia eran en escala de grises.

Para convertir una imagen PPM en escala de grises la matriz debe ser multiplicada por valores específicos según el pixel RGB.

Los pixeles rojos (R) deben ser multiplicados por 0.299, los píxeles verdes

(G) deben ser multiplicados por 0.587 y los azules (B) deben ser multiplicados por 0.114. El algoritmo utilizado se puede observar en el ***Anexo 2.***

Imagen 2. *Filtro gris aplicado en imagen de Lenna.*

c) Flipped and Flopped:

Los métodos flipped y flopped buscan dar tener en la imagen un efecto espejo ya sea en sentido vertical u horizontal, este efecto suele ser usado comúnmente en el diseño gráfico para la orientación que debe tener una imagen, esta orientación suele notarse en la mirada de un animal o persona en una imagen así como en objetos específicos.

Flopped busca realizar el efecto espejo en la imagen con base al eje vertical, para esto se debe garantizar que el último valor de la última columna en cada una de las filas sea el primero en la imagen resultante, el algoritmo utilizado se puede observar en el ***Anexo 3.***

Imagen 3. *Método flopped aplicado en imagen de Lenna.*

Flipped busca realizar el efecto espejo en la imagen con base en el eje horizontal, para esto se debe garantizar que la última fila sea la primera en la imagen resultante el algoritmo utilizado se puede observar en el ***Anexo 4.***

Imagen 4. *Método flipped aplicado en imagen de Lenna.*

d) Negativo:

El negativo de una imagen consiste principalmente en invertir los niveles de intensidad de la misma, de manera que la intensidad resultante en la imagen de salida disminuya conforme a la imagen de entrada.

Para esto se debe conocer el máximo valor

posible que puede tomar la imagen conocido también como número mágico y restarle el pixel actual de la imagen que debe ser cambiado, este proceso se debe repetir para toda la matriz de la imagen a aplicar el filtro.

$$inverso = máximo\ valor - pixel\ actual$$

El algoritmo utilizado se puede observar en el ***Anexo 5.***

Imagen 4. *Filtro negativo aplicado sobre imagen de Lenna.*

e) Colorize:

Este filtro busca poder llevar una imagen de la escala de grises a color, para llevarlo a cabo se hace uso de la técnica del pseudo color conocida también como Intensity Slicning Normalizada, para determinar los colores con que será reemplazada la escala de grises se hace uso de las intensidades del espectro de luz visible en un intervalo de 13 colores. Comúnmente sus usos son para la identificación de temperaturas en objetos y animales. ***Ver anexo 6.***

Imagen 5. *Filtro colorize aplicado sobre imagen de Lenna.*

f) Threshold:

El filtro threshold es una de las herramientas más sencillas y utilizadas para la segmentación de imágenes. Para cada pixel en la imagen si este es mayor que un valor especifico de threshold conocido también como valor de umbral el pixel es convertido a blanco, en caso contrario es convertido a negro.

El valor umbral es determinado a partir del Isodata (Iterative Self-Organizing Data Analysis) es una técnica simple iterativa desarrollada por Ridler y Calvard en 1978. El objetivo principal del isodata es dividir el histograma en dos partes las cuales representan dos sub-regiones de la imagen, este método utiliza una serie de pasos que permiten obtener el valor de umbral

1. Se debe elegir un valor inicial de threshold ***T.***
2. Se debe dividir la imagen en dos grupos R1 y R2 usando ***T***.
3. Se determinan μ1 y μ2

$$\mu 1(T) = \sum_{i=0}^{T} i * h(i) / \sum_{i=0}^{T} h(i)$$

$$\mu 2(T) = \sum_{i=T+1}^{maxValue} i * h(i) / \sum_{i=T+1}^{maxValue} h(i)$$

4. Se determina el nuevo valor de umbral T

$$T = (\mu 1 + \mu 2)/2$$

5. Se deben repetir los pasos 2-4 hasta que el valor de T no varíe.

Para ver el algoritmo implementado revise el ***Anexo 7.***

Imagen 5. *Filtro threshold aplicado sobre imagen de Lenna.*

g) Erosión:

El método de erosión es una de las operaciones morfológicas implementadas en el programa, esta suele utilizar un elemento estructurador para sondear y reducir las formas contenidas en una imagen, la intensidad de la erosión está determinada por un número entero, a mayor valor habrá mayor erosión.

Imagen 6. *Imagen original sin ningún filtro aplicado.*

Imagen 7. *Imagen después de haber sido aplicado el método de erosión.*

Para poder aplicar el método de erosión debe aplicarse el filtro de threshold, el algoritmo implementado se puede ver en el ***Anexo 8.***

h) Dilatación:

El método de dilatación es una de las operaciones morfológicas implementadas en el programa, esta suele utilizar un elemento estructurador para sondear y reducir las formas contenidas en una imagen, la intensidad de la dilatación está determinada por un número entero, a mayor valor habrá mayor dilatación. El algoritmo implementado se puede observar en el ***Anexo 9.***

Imagen 8. *Imagen después de haber sido aplicado el método de erosión.*

i) Mean Blurr:

Este método aplicará un difuminado sobre la imagen por medio de la convolución de la matrix de la imagen con una máscara conocida también como kernel específico.

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Este kernel será aplicado sobre la imagen obteniendo como resultado una matriz 3x3 compuesta por píxeles de la imagen, seguido a este se obtendrá un valor representativo de la matrix que será dividido entre 9 y este será uno de los pixeles de mi imagen de salida, este proceso se repite hasta recorrer toda la imagen.

En el programa se ha implementado la posibilidad de aumentar la intensidad del blur dando un valor de entrada mayor a 1, para esto se ha hecho una iteración que repite el procedimiento cuantas veces sea necesario, para ver el código revisar el ***anexo 10.***

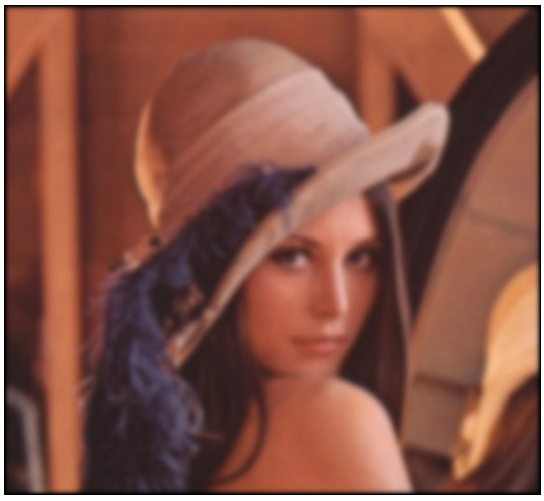

Imagen 9. *Filtro meanBlur aplicado sobre imagen de Lenna.*

j) Sharpen:

El método sharpen permite realzar los bordes de la imagen trabajando de una forma muy similar al meanBlurr este por medio de la convolución aplica el siguiente kernel sobre la imagen. El algoritmo se puede revisar en el ***anexo 11.***

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Esto da como resultado la siguiente imagen

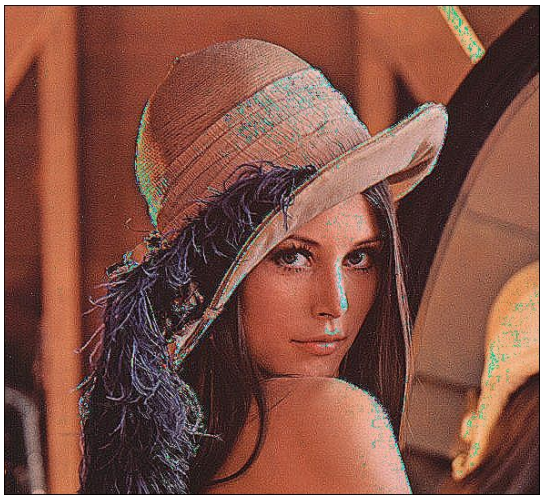

Imagen 10. *Filtro sharpen aplicado sobre imagen de Lenna.*

k) Combinación:

La combinación de imágenes es una de las herramientas más utilizadas en el diseño, se busca juntar dos imágenes en una sola donde la imagen resultante sea una con las dos imágenes en ella, para esto el objetivo principal es lograr la composición de mejor calidad posible y que el fondo de una de las imágenes sea eliminado por completo.

Por ejemplo, se busca poner la imagen de la ardilla con un chromakey sobre la plazoleta de ingenierías de la universidad del valle, la idea es que el resultado sea lo más preciso posible y el fondo verde de la imagen de la ardilla quede eliminado en su totalidad.

Imagen 11. *Ardilla univalle ChromaKey.*

Imagen 12. *Plazoleta de ingenierías universidad del valle.*

Imagen 13. *Composición entre imagen 11 y 12.*

El algoritmo utilizado se puede observar en el ***anexo 12.***

REFERENCIAS

[1] A HISTORY OF C++: 1979-1991 , Stroustrup Bjarne, Murray Hill, New Jersey.
[2] THRIVING IN A CROWDED AND CHANGING WORLD: C++ 2006 - 2020 , Stroustrup Bjarne, Morgan Stanley and Columbia University, USA.
[3] Procesamiento de Imágenes Tesis, Nicolás Aguirre Dobernack.
[4] Machine Learning and My Learning, “Pseudo Color Image Processing”. [En línea] Machine Learning and My Learning
[5] Universitat De Ovalencia, “Pseudo color y falso color” [En línea] Universitat De Ovalencia
[6] A C++ IMPLEMENTATION OF OTSU’S IMAGE SEGMENTATION METHOD, 2016, Balarini Juan Pablo, Nesmachnow Sergio, Universidad de la República, Facultad de Ingeniería, URUGUAY.
[7] USC Student Computing Facility, “CS 103 Chroma Key” [En línea] SCF USD EDU
[8] towards data science, “Depth Estimation: Basics and intuition” [En línea] Towards Data Science
[9] ARITIFICIAL INTELLIGENCE II “Computer Vision”, Olga Veksler [En línea] Computer Vision
[10] The university of Auckland New Zealand, “Morphological Image Processing” [En línea] Morphological Image Processing
[11] TÉCNICAS DE CLASIFICACIÓN AUTOMÁTICA DE USO DE SUELOS AGRÍCOLAS Y FORESTALES BASADAS EN IMÁGENES DIGITALES, 2013, Cruz Macedo Antonia, Peñas Santos Matilde, Martinsanz Pajares Gonzalo, Universidad Complutense de Madrid, Facultad de ciencias físicas, Madrid.

Anexos

a) Sepia

```
void ImagePPM::sepia(void){

  for(int i=0; i<red.size(); i++){
    int tr = round( 0.393*red[i] + 0.769*green[i] + 0.189*blue[i] );
    int tg = round( 0.349*red[i] + 0.686*green[i] + 0.168*blue[i] );
    int tb = round( 0.272*red[i] + 0.534*green[i] + 0.131*blue[i] );

    unsigned int r = (tr>255) ? 255 : tr;
    unsigned int g = (tg>255) ? 255 : tg;
    unsigned int b = (tb>255) ? 255 : tb;

    red[i]= r;
    green[i]= g;
    blue[i]= b;
  }
}
```

Anexo 1. *Código fuente de filtro sepia.*

b) Gris:

```
void ImagePPM::grayScale(void){

  multiplyVectorBy(0.299, red);
  multiplyVectorBy(0.587, green);
  multiplyVectorBy(0.114, blue);

  for(int i=0; i<red.size(); i++){
    unsigned int temp = red[i]+green[i]+blue[i];
    red[i]= temp;
    green[i]= temp;
    blue[i]= temp;
  }
}
```

Anexo 2. *Código fuente de filtro Gris.*

c) Flipped and Flopped:

```
void ImagePPM::flopped(void){

  std::vector<unsigned int> tempRed;
  std::vector<unsigned int> tempGreen;
  std::vector<unsigned int> tempBlue;

  tempRed.reserve( red.size() );
  tempGreen.reserve( green.size()) ;
  tempBlue.reserve( blue.size() );


  for(int i=0; i<rows; i++){
    for(int j=0; j<cols; j++){

      tempRed.push_back( red[ (cols*(i+1)) - 1 - j ] );
      tempGreen.push_back( green[ (cols*(i+1)) - 1 - j ] );
      tempBlue.push_back( blue[ (cols*(i+1)) - 1 - j ] );

    }
  }

  for(int i=0; i<red.size(); i++){
    red[i] = tempRed[i];
    green[i] = tempGreen[i];
    blue[i] = tempBlue[i];
  }

}
```

Anexo 3. *Código fuente de flopped.*

```
void ImagePPM::flipped(void){

  std::vector<unsigned int> tempRed;
```

```
  std::vector<unsigned int> tempGreen;
  std::vector<unsigned int> tempBlue;

  tempRed.reserve( red.size() );
  tempGreen.reserve( green.size()) ;
  tempBlue.reserve( blue.size() );


  for(int i=rows; i>0; i--){
    for(int j=0; j<cols; j++){

      tempRed.push_back( red[ (cols*(i-1)) + j ] );
      tempGreen.push_back( green[ (cols*(i-1)) + j ] );
      tempBlue.push_back( blue[ (cols*(i-1)) + j ] );

    }
  }

  for(int i=0; i<red.size(); i++){
    red[i] = tempRed[i];
    green[i] = tempGreen[i];
    blue[i] = tempBlue[i];
                                  }

                                 }
```

Anexo 4. *Código fuente de flipped.*

d) Negativo:

```
void ImagePPM::negative(void){

    for(int i = 0; i < red.size(); i++){

        red[i] = maxVal - red[i];
```

```
        green[i] = maxVal - green[i];
        blue[i] = maxVal - blue[i];
    }
}
```

Anexo 5. *Código fuente de negativo.*

e) Colorize:

```
void ImagePPM::colorize(void){

  /**
   * the image has to be converter into grayscale so that
   * way we can assign of the false color to the image
  */
  grayScale();

  int max=0, min=255;
  for(int i=0; i<red.size(); i++){ //finds the min ans max
intensities
    if(red[i] > max)
      max = red[i];
    if(red[i] < min)
      min = red[i];
  }

  // to understand this below part check "intensity slicing"
  // this is the method use to assign the pseudocolor
  int planeDistance, planes[14];
  planeDistance = round( (max-min)/13.0 );


  planes[0]=min;
  planes[14-1]=max;
  for(int i=1; i<13; i++){
    planes[i] = planes[i-1] + planeDistance;
  }
```

```
for(int i=0; i<red.size(); i++){

  if( planes[0] <= red[i] && red[i] < planes[1] ){
    red[i] = 148;
    green[i] = 0;
    blue[i] = 211;
  }
  else if( planes[1] <= red[i] && red[i] < planes[2] ){
    red[i] = 75;
    green[i] = 0;
    blue[i] = 130;
  }
  else if( planes[2] <= red[i] && red[i] < planes[3] ){
    red[i] = 38;
    green[i] = 0;
    blue[i] = 193;
  }
  else if( planes[3] <= red[i] && red[i] < planes[4] ){
    red[i] = 0;
    green[i] = 0;
    blue[i] = 255;
  }
  else if( planes[4] <= red[i] && red[i] < planes[5] ){
    red[i] = 9;
    green[i] = 80;
    blue[i] = 247;
  }
  else if( planes[5] <= red[i] && red[i] < planes[6] ){
    red[i] = 9;
    green[i] = 247;
    blue[i] = 235;
  }
```

```
    else if( planes[6] <= red[i] && red[i] < planes[7] ){
      red[i] = 0;
      green[i] = 255;
      blue[i] = 0;
    }
    else if( planes[7] <= red[i] && red[i] < planes[8] ){
      red[i] = 127;
      green[i] = 255;
      blue[i] = 0;
    }
    else if( planes[8] <= red[i] && red[i] < planes[9] ){
      red[i] = 255;
      green[i] = 255;
      blue[i] = 0;
    }
    else if( planes[9] <= red[i] && red[i] < planes[10] ){
      red[i] = 255;
      green[i] = 191;
      blue[i] = 0;
    }
    else if( planes[10] <= red[i] && red[i] < planes[11] ){
      red[i] = 255;
      green[i] = 127;
      blue[i] = 0;
    }
    else if( planes[11] <= red[i] && red[i] < planes[12] ){
      red[i] = 255;
      green[i] = 68;
      blue[i] = 0;
    }
    else if( planes[12] <= red[i] && red[i] < planes[13] ){
      red[i] = 255;
      green[i] = 0;
      blue[i] = 0;
```

```
    }
  }
}
```

Anexo 6. *Código fuente de colorize.*

f) Threshold:

```
void ImagePPM::threshold(void){

  if(vectorHistogramaGrises.empty()){
    grayScale();
    histogramaGrises();
  }

  double miu1, miu2, numerador =0, demominador =0;
  bool flag = true;
  double T_i = 127, T_f = 0, thresholdValue=0;

  while (flag)
  {

    for(int i=0; i<=T_i; i++){
    numerador += i * int(vectorHistogramaGrises[i]);
    demominador += int(vectorHistogramaGrises[i]);
    }

    miu1 = numerador/demominador;

    numerador=0; demominador=0;
    for(int i=T_i; i<=255; i++){
      numerador += i * int(vectorHistogramaGrises[i]);
      demominador += int(vectorHistogramaGrises[i]);
    }

    miu2 = numerador/demominador;
```

```
    double T = (miu1+miu2)*0.5;

    T_f = T;
    if(T_i == T_f){
      thresholdValue = round( T_f );
      flag = false;
    }else{
      T_i = T;
    }

  }


  for(int i=0; i<red.size(); i++){
    if( red[i] < thresholdValue ){
      red[i] = 0;
    }
    else if(red[i] >= thresholdValue){
      red[i] = 255;
    }
  }

  green = red;
  blue = red;

}
```

Anexo 7. *Código fuente de threshold.*

g) Erosión:

```
void ImagePPM::erosion(int num){

  try{
    if(3>num){
      throw "The input number has to be greater than 3";
```

```
    }else if(num%2 == 0){
      throw "The input number has to be an odd number";
    }
    else if(num < 0){
      throw "The input number has to be a positive integer";
    }
    else if(cols<=num || rows<=num){
      throw "The input number is too big";
    }

  }catch(const char* e){
    std::cout<< e <<"\n";
    exit(EXIT_FAILURE);
  }
  threshold();

  /**
    * mask is the num x num matrix, wich is going to be using to
compare
   * a specific grid of values with the rest of the image */
  int maskCenterRow = floor( num/2.0 );
  int maskCenterCol = floor( num/2.0 );

  std::vector<unsigned int> tempRed (red.size(), 0);
  std::vector<unsigned int> tempGreen (green.size(), 0);
  std::vector<unsigned int> tempBlue (blue.size(), 0);

  tempRed = red;
  tempGreen = green;
  tempBlue = blue;

  for(int i= maskCenterRow; i<(rows-maskCenterRow); i++){
    for(int j= maskCenterCol; j<(cols-maskCenterCol); j++){
```

```
      if( red[(cols*i) + j] == 0 ){

        for(int k=i-maskCenterRow; k<(i-maskCenterRow)+num; k++){
                for(int l=j-maskCenterCol; l<(j-maskCenterCol)+num;
l++){

              tempRed[(cols*k)+l] = 0;
              tempGreen[(cols*k)+l] = 0;
              tempBlue[(cols*k)+l] = 0;
          }
        }
      }

    }
  }

  red.clear();
  green.clear();
  blue.clear();

  red = tempRed;
  green = tempGreen;
  blue = tempBlue;

}
```

Anexo 8. *Código fuente de erosion.*

h) Dilatación:

```
void ImagePPM::dilation(int num){

  try{
    if(3>num){
      throw "The input number has to be greater than 3";
    }else if(num%2 == 0){
      throw "The input number has to be an odd number";
```

```
    }
    else if(num < 0){
      throw "The input number has to be a posite integer";
    }
    else if(cols<=num || rows<=num){
      throw "The input number is too big";
    }

  }catch(const char* e){
    std::cout<< e <<"\n";
    exit(EXIT_FAILURE);
  }


  threshold();

  /**
    * mask is the num x num matrix, wich is going to be using to
compare
   * a specific grid of values with the rest of the image */
  int maskCenterRow = floor( num/2.0 );
  int maskCenterCol = floor( num/2.0 );

  std::vector<unsigned int> tempRed (red.size());
  std::vector<unsigned int> tempGreen (green.size());
  std::vector<unsigned int> tempBlue (blue.size());

  tempRed = red;
  tempGreen = green;
  tempBlue = blue;


  for(int i= maskCenterRow; i<(rows-maskCenterRow); i++){
    for(int j= maskCenterCol; j<(cols-maskCenterCol); j++){
```

```
      if( red[(cols*i) + j] == 255 ){

        for(int k=i-maskCenterRow; k<(i-maskCenterRow)+num; k++){
                for(int  l=j-maskCenterCol;  l<(j-maskCenterCol)+num;
l++){

            tempRed[(cols*k)+l] = 255;
            tempGreen[(cols*k)+l] = 255;
            tempBlue[(cols*k)+l] = 255;

          }
        }
      }

    }
  }

  red.clear();
  green.clear();
  blue.clear();

  red = tempRed;
  green = tempGreen;
  blue = tempBlue;

}
```

Anexo 9. *Código fuente de dilatacion.*

i) meanBlurr:

```
void ImagePPM::meanBlur(int  intensityLevel){

  double kernel[3][3] = { {1,1,1},
```

```
                          {1,1,1},
                          {1,1,1}
                        };

  for(int i=0; i<intensityLevel; i++){
    std::vector <int> tempRed;
    std::vector <int> tempGreen;
    std::vector <int> tempBlue;

    convolution(kernel, tempRed, tempGreen, tempBlue);

    double num = (1.0/9.0);

    multiplyVectorBy(num, tempRed);
    multiplyVectorBy(num, tempGreen);
    multiplyVectorBy(num, tempBlue);

    for (int i=0; i<red.size(); i++){
      red[i] = tempRed[i];
      green[i] = tempGreen[i];
      blue[i] = tempBlue[i];
    }

  }

}
```

Anexo 10. *Código fuente de meanBlurr.*

j) Sharpen:

```
void ImagePPM::sharpen(void){

  double kernel[3][3] = { {0,-1,0},
                          {-1,5,-1},
                          {0,-1,0}
                        };
```

```
    std::vector <int> tempRed;
    std::vector <int> tempGreen;
    std::vector <int> tempBlue;

    convolution(kernel, tempRed, tempGreen, tempBlue);

    for (int i=0; i<red.size(); i++){
      red[i] = int( std::abs( tempRed[i] ) );
      green[i] = int( std::abs( tempGreen[i] ) );
      blue[i] = int( std::abs( tempBlue[i] ) );
    }

}
```

Anexo 11. *Código fuente de Sharpen.*

k) Combinación:

```
void ImagePPM::chromaKey(ImagePPM &other){

  try{
    if(other.rows > rows  ||  other.cols > cols){
      throw "The image is too big";
    }
  }catch(const char* e){
    std::cout<< e <<"\n";
  }


    int  chromaRed  =  round(  (  other.red[0]  +  other.red[1]  +
other.red[other.cols]    +    other.red[other.cols    -    1]    +
other.red[other.cols  -  2]  +  other.red[(2*other.cols)  -  1]  +
other.red[(other.rows  -  1)*other.cols]  +  other.red[(other.rows  -
1)*other.cols + 1] ) / 8.0 );

   int  chromaGreen  =  round(  (  other.green[0]  +  other.green[1]  +
```

```
other.green[other.cols]    +    other.green[other.cols    -    1]    +
other.green[other.cols  -  2]  +  other.green[(2*other.cols)  -  1]  +
other.green[(other.rows  -  1)*other.cols] + other.green[(other.rows
- 1)*other.cols + 1] ) / 8.0 );

    int  chromaBlue  =  round(  (  other.blue[0]  +  other.blue[1]  +
other.blue[other.cols]    +    other.blue[other.cols    -    1]    +
other.blue[other.cols  -  2]  +  other.blue[(2*other.cols)  -  1]  +
other.blue[(other.rows  -  1)*other.cols] + other.blue[(other.rows -
1)*other.cols + 1] ) / 8.0 );



        int     foregroundBegin_row     =     floor(     ((     1     -
other.rows/double(rows))/2.0) * rows ) ;
        int     foregroundBegin_col     =     floor(     ((     1     -
other.cols/double(cols))/2.0) * cols ) ;



  for(int i=0; i<other.red.size(); i++){

         if(   (chromaGreen   -   10)   <=   int(other.green[i])     &&
int(other.green[i]) <= (chromaGreen + 10) ){

            if(   (chromaBlue   -   10)   <=   int(other.blue[i])     &&
int(other.blue[i]) <= (chromaBlue + 10) ){

               if(   (chromaRed   -   10)   <=   int(other.red[i])     &&
int(other.red[i]) <= (chromaRed + 10) ){
          other.red[i] = 0;
          other.green[i] = 255;
          other.blue[i] = 0;
        }
      }
```

```
    }

  }

  std::vector<unsigned int> tempRed ( red.size() );
  std::vector<unsigned int> tempGreen ( green.size() );
  std::vector<unsigned int> tempBlue ( blue.size() );

  tempRed = red;
  tempGreen = green;
  tempBlue = blue;

  int contador=0;
    for(int  i=foregroundBegin_row;  i  <  (foregroundBegin_row  +
other.rows); i++){
      for(int  j=foregroundBegin_col;  j  <  (foregroundBegin_col  +
other.cols); j++){

      red[(cols*i) + j] = other.red[contador];
      green[(cols*i) + j] = other.green[contador];
      blue[(cols*i) + j] = other.blue[contador];

      contador++;

      }
    }


  for(int i=0; i<red.size(); i++){

    if( int(green[i]) == 255 ){

      if( int(blue[i]) == 0 ){
```

```
        if( int(red[i]) == 0 ){

          red[i] = tempRed[i];
          green[i] = tempGreen[i];
          blue[i] = tempBlue[i];
        }
      }
    }

  }


}
```

Anexo 12. *Código fuente de Composición.*