EIE
Escuela de
Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

UNIVERSIDAD DE COSTA RICA

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERIA ELÉCTRICA

ESTRUCTURAS ABSTRACTAS DE DATOS Y ALGORITMOS PARA INGENIERÍA

INFORME PROYECTO 0: EFÉCTOS EN IMÁGENES DIGITALES

ESTUDIANTES:
JESÚS ZÚÑIGA MÉNDEZ (B59084)
DENNIS CHAVARRÍA SOTO (B82097)

PROFESOR:
RICARDO ROMÁN BRENES; M. SC.

II CICLO 2019

Índice

Índice de figuras **1**

1. Reseña del programa **2**

2. Funcionamiento del programa **2**
2.1. Filtros . . . 3
2.1.1. Filtro Gaussiano . . . 3
2.1.2. Filtro Desviación estandar . . . 4
2.1.3. Detección de bordes . . . 5
2.1.4. Difuminado de movimiento . . . 5
2.1.5. Ruido sal y pimienta (S&P) . . . 5
2.1.6. Erosión y Dilatación . . . 5
2.1.7. Filtro Inversión de Color . . . 6
2.1.8. Filtro Transformación a escala de grises . . . 6
2.1.9. Main . . . 6

3. Experimentos Realizados **6**

4. Resultados **8**

5. Conclusiones **13**

6. Referencias **14**

7. Anexos **15**

Índice de figuras

1. Foto Original . . . 8
2. Foto con el filtro Gaussiano . . . 8
3. Foto con difuminado de movimiento . . . 9
4. Foto con ruido sal-pimienta . . . 9
5. Foto en blanco y negro . . . 10
6. Foto con inversión de colores . . . 10
7. Foto con erosión . . . 11
8. Foto con dilatación . . . 11
9. Foto con detector de bordes . . . 12
10. Foto con filtro de desviación estándar . . . 12

1. Reseña del programa

Como parte del plan del curso de Estructuras Abstractas y Algoritmos para Ingeniería, es necesario desarrollar un primer proyecto, que permite a los estudiantes aplicar los conocimientos adquiridos de programación hasta entonces, para solucionar un problema dado. En esta oportunidad, el objetivo fue el de desarrollar un programa que tenga la capacidad de aplicar filtros morfológicos a imágenes que el usuario selecciona, para, luego, devolver una copia de la misma pero modificada.

El programa fue desarrollado con el lenguaje de programación C++, e implementa la librería OpenCV; tiene la capacidad de ser invocado mediante la línea de comandos, y buscar el archivo de imagen y el filtro que el usuario indique. Cuando esto ocurre, se realizan las operaciones pertinentes para editar el archivo; una vez resuelto, se exporta el resultado final con el formato de salida que el usuario indica al invocar la aplicación.

El impacto en la formación profesional y académica de los estudiantes, al tener que desarrollar un software que complete el objetivo establecido por el profesor, es inmedible en cuanto, entre otros aspectos, aumenta la capacidad de razonamiento y análisis para solución de problemas; además de aprender a utilizar librerías de terceros con métodos, variables y funcionalidades totalmente distintas, pero versátiles, que no estaban disponibles anteriormente.

2. Funcionamiento del programa

El programa se desarrolló con el lenguaje de programación C++, sin embargo, para la lectura y escritura se hace uso de la librería OpenCV. Las imágenes objetivo deben estar en la carpeta del programa y este se invoca siguiendo la sintaxis *./proyecto0 IMAGEN.FORMATO OPERACION NUEVOFORMATO* de forma que, el nombre del ejecutable es proyecto0; IMAGEN.FORMATO es el objetivo de la edición, donde el nombre y su extensión son los del archivo original, la operación debe ser una de las siguientes:

- FG: Filtro Gaussiano, hace ver borrosa la imagen.
- FSTD: Filtro de Desviación Estándar, asigna la desviación estándar a los pixeles de la imagen, remarcando los borden y coloreando lo demás con colores muy oscuros.
- ED: Detección de bordes (edge detection), resalta los bordes de la imagen en blanco y negro
- MB: Difuminado de movimiento (motion blur), crea el efecto de desenfoque de una imagen tomada por una cámara en movimiento
- SP: Ruido sal y pimienta, añade ruido a la imagen, lo que crea el efecto de que se añade sal y pimienta
- E: Erosión. Reduce los bordes en blanco y negro de una imagen, parece que la reduce o que erosiona los bordes.

- (D) Dilatación, este efecto crea el resultado opuesto al de erosión, aumentando bordes y franjas de color, de forma que parece aumentar el tamaño de los objetos.
- (I) Inversión de color consiste en que la imagen se re coloree con colores negativos.
- (G) Transformación de escala de grises, transforma una imagen a color, de tres canales, a una en blanco y negro.

Seguido a la operación que se desea aplicar, se indica el forma de salida de la imagen. La edición genera un archivo llamado o *IMAGEN.EXTENSION.OPERACION.NUEVOFORMATO*. Esto debe repetirse de la misma forma varias veces si se desea aplicar varias ediciones a la foto, o distorsionarla más con los filtros gaussiano o de movimiento.

2.1. Filtros

Para todos los filtros se utilizan objetos de tipo MAT, tipo que se incluye en la librería OpenCV. Este consiste en una matriz multidimensional que almacena los valores de intensidad para cada canal, en caso de ser de 3 colores, o de un solo canal, si es en escala de grises. Se implementa un MAT imagen para la fotografía que es leída, y un MAT nueva para la imagen editada.

2.1.1. Filtro Gaussiano

Este filtro produce el efecto de distorsionado en imágenes mediante la asignación del valor promedio de los valores de intensidad de cada pixel de un kernel a uno central, esto para cada canal si es a color (universidad de Murcia, s.f.). Una de sus utilidades es la de eliminar ruido en imágenes. Para implementar el código se defininen 3 vectores, del tipo Vec3b que incluye la librería OpenCV.

- Tripleta1 almacena los valores de pixel tomados de la matriz de la imagen.
- Pixel es un vector de 3 canales que se utiliza para asignar a la matriz de la nueva imagen editada los valores de pixel modificados. Los guarda en el orden azul, verde y rojo.
- TriValoresPixel almacena 3 valores de pixel que corresponden a los promedios que el filtro Gaussiano debe obtener. Son 3 números que corresponden a cada canal de la imagen; azul, verde y rojo, en tal orden.
- fprom es una variable de tipo float que almacena los valores de la suma de la intensidad de los pixeles del kernel.
- iprom es una variable de tipo int que almacena los valores de fprom después de que este obtuviera la suma de las intensidades en el kernel pero los divide entre 8 y lo convierte a un número entero; de esta forma es almacenable en los vectores Vec3b de OpenCV.

La estructura de este método consiste en un conjunto de ciclos anidados de tipo ”for”. El primero de ellos itera por cada fila, luego uno interno itera por columna, seguido hay uno que repite tres veces, para cambiar el color; este tiene la particularidad de que su valor de repetición se utiliza en los vectores para asignar el los promedios iprom a una posición determinada.

2.1.2. Filtro Desviación estandar

Este filtro crea el efecto de pintar la pantalla de color negro y resaltar mucho los bordes de la imagen original, lo que lo hace parecer un detector de borde; esto mediante la asignación del valor de desviación estándar obtenido por los valores de intensidad de cada pixel de un kernel a uno central (Mathworks, s.f.-b)., esto para cada canal si es a color. Una de sus utilidades es la de eliminar ruido en imágenes. Para implementar el código se defininen 3 vectores, del tipo Vec3b que incluye la librería OpenCV.

- Tripleta1 almacena los valores de pixel tomados de la matriz de la imagen.
- Pixel es un vector de 3 canales que se utiliza para asignar a la matriz de la nueva imagen editada los valores de pixel modificados. Los guarda en el orden azul, verde y rojo.
- TriValoresPixel almacena 3 valores de pixel que corresponden a los promedios que del kernel. Son 3 números que corresponden a cada canal de la imagen; azul, verde y rojo, en tal orden.
- fprom es una variable de tipo float que almacena los valores de la suma de la intensidad de los pixeles del kernel.
- iprom es una variable de tipo int que almacena los valores de fprom después de que este obtuviera la suma de las intensidades en el kernel pero los divide entre 8 y lo convierte a un número entero; de esta forma es almacenable en los vectores Vec3b de OpenCV.
- Distancia es una variable de tipo int que se utiliza para almacenar las sumas elevadas al cuadrado de la ecuación de la desviación estándar

Ya se mencionó la existencia de un ciclo que itera por cada color, sin embargo, dentro de él hay dos repetidores más además de un condicional. Primeramente, el valor de fprom se establece como 0 cada vez que se entra a este contexto, debido a que el valor de promedio del nuevo canal es diferente. Luego se ingresa al primer ciclo que empieza en la posición x=-1 y termina en x=1. Este se encarga de repetir por cada fila del kernel, luego hay otro contexto de repetición que tiene el mismo rango que el anterior, pero su variable es llamada *rep*. La finalidad de estos dos últimos *for* es definir el tamaño del kernel en el cuál se obtendrá el promedio de los valores de intensidad de los pixeles. El condicional se encarga de omitir el pixel central del kernel, además de los valores de los extremos de la matriz de la imagen; así evita un core dumped. Luego de obtener la suma, se guardan como promedios por canal en iprom, este se usa para asignarlos a TriPromediosCanal.

Hasta aquí, el método del filtro de la desviación estándar es idéntico al del filtro gaussiano, sin embargo, ahora se tiene otros dos ciclos que usan la variable x y rep con los mismos rangos anteriormente mencionados, así como la inclusión del mismo condicional. Dentro de todo se encuentra tripleta1, que tomará los valores de la matriz de imagen original y seleccionará una de las intensidades de algún color, según el valor de la variable del iterador de color; así, se restará al valor del promedio que corresponde al canal y se elevará al cuadrado. Tal resultado corresponde a distancia, el cuál se sumará por cada pixel de la matriz original.

El valor que tomará el pixel se asigna inmediatamente antes de iterar por cada color, así se llena la tripleta y, antes de cambiar de pixel desde el que se toma el kernel, se asignan los tres valores de pixel a la nueva matriz de la imagen que se creará.

2.1.3. Detección de bordes

La detección de bordes se basa en el algoritmo de Canny, primero se convierte la imagen a escala de grises tal y como se realiza en el filtro de escala de grises de este apartado, después se aplica el filtro Gaussiano para eliminar el ruido de la imagen y por ultimo se aplica la operación de Kernel de Sobe, en la cual básicamente se realiza el calculo de la gradiente vertical y horizontal, después de esto se calcula la norma de las gradientes y en base a esto se definen los bordes, asignando a los pixeles los valores obtenidos. Es importante realizar una serie de filtrados para garantizar que el resultado sea el optimo, uno de ellos es comparar los pixeles que estén perpendiculares al angulo de la gradiente, si los pixeles de mayor valor se encuentran en dicha dirección se acepta ese pixel como un máximo si no se descarta.

2.1.4. Difuminado de movimiento

Para el difuminado de movimiento, el efecto es el de una foto tomada por una cámara en movimiento. Se definen 4 vectores Vec3b, lo cuáles son pixel, tripleta1, tripleta2 y tripleta 3. Luego, se definen tres variables de tipo int que son promedio-rojo, promedio-verde y promedio-azul. luego tiene un ciclo que repite 4 veces el procedimiento que se explicará a continuación. Dentro se encuentra el iterador de filas y el de columnas. Dentro del último contexto se define a pixel como el vector que contiene los valores de los 3 canales que conforman a la matriz de la imagen original localizado en las coordenadas i, j que los ciclos determinan. Sigue un condicional que solo funciona si no se encuentra en los bordes de la matriz, esto para evitar el core dumped. Posteriormente, cada tripleta obtiene los valores de la intensidad de los tres canales del los pixeles, central, izquierdo y derecho, de i,j; luego se calculan los promedios de cada color con las intensidades de cada tripleta, una vez hecho esto, se asignan a una posición del vector pixel para luego asignarlo a la nueva matriz de imagen. Este procedimiento es el que se repite por 4 veces, para obtener el efecto de difuminado.

2.1.5. Ruido sal y pimienta (S&P)

En este filtro se agregan pixeles blancos y negros de forma aleatoria a los pixeles de la imagen, es un filtro relativamente sencillo ya que depende de un numero aleatorio y de un porcentaje de ruido que se quiere agregar, se recorre la matriz de la imagen en orden y cada cierta cantidad de pixeles se decide si se modifica el pixel o no, en este filtro es importante variar el grado de desplazamiento de los pixeles a modificar para evitar que se creen lineas verticales u horizontales poco estéticas en la imagen

2.1.6. Erosión y Dilatación

Estos filtros se comportan exactente igual, la única diferencia entre ellos es si se asigna un pixel negro o blanco, y su elemento estructuraste, un elemento estructuraste se una matriz de algún tamaño en este caso 3 x 3, en esta matriz se agregan valores de estructuración 0 o 1 ya que es una operación morfológica sobre imágenes binarias, se compara el vecindario de el pixel estudiado contra el elemento estructuraste, en el caso de la dilatación ningún pixel vecino debe ser mayor a la matriz de ceros y en el caso de la erosión ninguno debe ser menor a la matriz de unos, si se cumple la condición se mantiene el pixel si no se cambia por el valor contrario, el grado del filtro se puede cambiar modificando las dimensiones de el elemento estructuraste,

es importante aclarar que este filtro puede ser aplicado a imágenes a color, sin embargo la operación morfológica fue la primera utilizada y para efectos de este proyecto se quiso utilizar sobre imágenes binarias para ver un resultado mas marcado en cada imagen.

2.1.7. Filtro Inversión de Color

Este filtro invierte los colores, comúnmente llamados colores negativos. Define el vector de tipo Vec3b, pixel; también un número entero llamado PixelTemporal. Utiliza un ciclo para iterar entre columnas, dentro tiene otro que itera entre filas y, en ese último contexto, guarda a pixel el valor de la matriz en las coordenadas i, j que dependen de los primeros dos ciclos. Luego, repite 3 veces el procedimiento de asignar a pixel temporal el valor que guarda el vector en el primer canal, el azul, luego a pixel le asigna el valor de la variable temporal pero siendo restada a 255. De esta forma se repite para cada color; el nuevo vector de 3 canal se asigna a la nueva matriz de imagen, luego se repite el proceso para cada uno de la matriz real hasta recrear la imagen final editada.

2.1.8. Filtro Transformación a escala de grises

Este filtro permite transformar una imagen a color a una en blanco y negro. define el vector de tipo Vec3b, pixel; también los enteros B, G, R y ValorTriMedio. Utiliza un ciclo para iterar entre columnas, dentro tiene otro que itera entre filas y, en ese último contexto, guarda a pixel el valor de la matriz en las coordenadas i, j que dependen de los primeros dos ciclos. Luego, a las variables B, G y R asigna los valores de las posiciones 0, 1, 2 del vector pixel. ValorTriMedio guarda el promedio entre los valores de las 3 variables anteriores; luego, este resultado se asigna a cada posición del vector pixel, osea a cada canal; posteriormente a la matriz de nueva imagen, se le asigna al pixel que coincide con la posición i,j de los ciclos, los valores de pixel.

2.1.9. Main

El método main es el encargado de operar con los objetos como filtros e imágenes. Se encarga de procesar la entrada del usuario y asociarla con las operaciones, formatos e indicaciones adecuadas.

3. Experimentos Realizados

Como plataforma de creación y validación del programa se escogió C++ y junto al sistema operativo linux, esto gracias a que estas dos plataformas permiten un mayor desarrollo de los conocimientos gracias a su mayor nivel de complejidad.

Para la prueba y validación del programa que se realizó se hicieron múltiples corridas con cada uno de los filtros que se crearon, esto con el fin de realizar ajustes oportunos para la optimización de cada filtro, Además se corrieron diferentes pruebas del programa para cada filtro con la misma imagen y con imágenes distintas para poder cerciorarse que los filtros se aplicaran correctamente independientemente de imagen. A continuación se sintetizan las diferentes etapas de experimentación y sus resultados.

1. Se realizaron pruebas con opencv para corroborar que fuese capaz de leer y escribir en todos los formatos, como resultado se obtuvo que podía manejar todos los formatos excepto gif.

2. Se busco una alternativa para resolver el problema del gif, la solución fue usar imageMagick para leer el gif y realizar la conversión a un formato soportado por opencv
3. Se crearon filtros gradualmente, enfocándose en cada oportunidad en un solo filtro para garantizar que los resultados fueran aceptables con cada uno, se corría una prueba con cada filtro y se evaluaba la imagen haciendo ajustes en caso de ser necesario
4. Una vez que cada filtro estuvo optimizado, se realizaron múltiples corridas con diferentes tamaños de imagen para comprobar la robustez del programa escrito
5. Se intento optimizar el código para mejorar de alguna forma el tiempo de ejecución.

4. Resultados

De los experimentos realizados, se toma una imagen original, al aplicar las modificaciones que puede realizar el programa, se obtienen imágenes con aspecto distinto, acorde con la operación seleccionada, tal y como se muestra en las siguientes figuras:

 ;

Figura 1: Foto Original

 ;

Figura 2: Foto con el filtro Gaussiano

 ;

Figura 3: Foto con difuminado de movimiento

 ;

Figura 4: Foto con ruido sal-pimienta

 ;

Figura 5: Foto en blanco y negro

 ;

Figura 6: Foto con inversión de colores

 ;

Figura 7: Foto con erosión

 ;

Figura 8: Foto con dilatación

 ;

Figura 9: Foto con detector de bordes

 ;

Figura 10: Foto con filtro de desviación estándar

Es recalcable que la figura 8 aumenta la distribución del color negro en la imagen, eso hace que se noten menos los detalles faciales en los ojos, así como el pelo; por otra parte, la dilatación realiza el efecto contrario, por lo que se remarcan más zonas con el color blanco y los detalles faciales, principalmente los ojos, destacan más.

5. Conclusiones

- Se desarrolló un programa con la capacidad de aplicar múltiples filtros morfológicos que puede leer y escribir imágenes editadas a partir de las operaciones indicadas por el usuario.
- Todas las operaciones para la edición de las fotografías fueron implementadas satisfactoriamente.
- Fue posible la aplicación de librerías de terceros, en este caso, OpenCV para lograr el adecuado funcionamiento del programa.
- Queda patente el hecho de que las librerías de terceros facilitan la implementación y desarrollo de funcionalidades; también incluyen métodos que pueden reducir el código y tipos de variables que proporcionan aún más funcionalidad

6. Referencias

- Mathworks. (s.f.-a). Edge detection. Recuperado de: https://www.mathworks.com/discovery/edge-detection.html.
- Mathworks. (s.f.-b). stdfilt. Recuperado de: https://es.mathworks.com/help/images/ref/stdfilt.html.
- Murias, D. (2017). 16 fotos manipuladas que pasaron a la historia. Recuperado de: https://magnet.xataka.com/un-mundo-fascinante/16-fotos-manipuladas-que-pasaron-a-la-historia.
- universidad de Murcia. (s.f.). Tecnicas de filtrado. ´ Recuperado de: https://www.um.es/geograf/ sigmur/teledet/tema06.pdf. Universidad de Murcia.
- thedenverherald(2018). Rising star Who is Mackenzie Davis? New Terminator movie actress and star of Blade Runner Recuperado de: https://thedenverherald.com/rising-star-who-is-mackenzie-davis-new-terminator-movie-actress-and-star-of-blade-runner/

7. Anexos

main.cpp:

```
/**
 * @file main.c
 * @author Jesus Zu iga Mendez
 * @author Dennis Chavarria
 * @brief Archivo pricipal, Proyecto0 filtro en imagenes
 * @version 1.0
 * @date 26 de octubre de 2019
 * @copyright Copyleft (l) 2019
 */
#include "./include/Includes.hpp"

using namespace std;
using namespace cv;
using namespace Magick;
/**
 * @brief Funcion Principal
 */
int main(int argc, char** argv)
{
   if (argc == 4){
      string imagen = argv[1];
      string operacion = argv[2];
      string formato = argv[3];
      //string porciento = argv[4];
      //float porcientoto = stof(porciento);
      string extension = "";
      Filtros filtros(imagen);

      //comprobamos si la extension es gif en caso de serlo se guarda y cambia a bmp
      for(int i = imagen.size();i>-1; i--){
         if (((imagen[i] >= 97) && (imagen[i] <= 122)) || ((imagen[i] >= 65) && (imagen[i] <= 90))) {
            extension = extension + imagen[i];
         }else if(imagen[i] == '.') {
            i = -1;
         }
      }
      //si es gif la convertimos a bmp y trabajamos sobre esa
      if ((extension == "fig") || (extension == "FIG")){
         //codigo tomado de https://imagemagick.org/script/magick++.php
         InitializeMagick(*argv);
         // Construct the image object. Seperating image construction from the
         // the read operation ensures that a failure to read the image file
         // doesn't render the image object useless.
         Image image;
         try {
           // Read a file into image object
           image.read(imagen);
           // Crop the image to specified size (width, height, xOffset, yOffset)
           //image.crop( Geometry(250,250));//, 100, 100) );
           // Write the image to a file
           image.write("TempConvertGifTo.bmp");
           Mat abrirImagen = imread("TempConvertGifTo.bmp",CV_LOAD_IMAGE_COLOR);
           filtros.matrizImagen = abrirImagen;
           int s = system("rm TempConvertGifTo.bmp");
```

```
55            s=s;
56          }
57          catch( Magick::Exception &error_ )
58          {
59            cout << "Caught exception: " << error_.what() << endl;
60            return 1;
61          }
62        }else{
63        }
64        //
65        //
66        //
67        if (operacion == "FG"){
68          //(FG) Filtro gaussiano.
69          cout << "Gausiano" << endl;
70          filtros.FiltroGaussiano();
71        }else if (operacion == "FSTD")
72        {
73          //(FSTD) Filtro de desviaci n est ndar.
74          cout << "Desviacion" << endl;
75          filtros.FiltroDesviacionEstandar();
76        }else if (operacion == "ED")
77        {
78          //(ED) Detecci n de bordes (edge detection).
79          cout << "Deteccion de bordes" << endl;
80          filtros.TransformacionEscalaGrises();
81          filtros.FiltroGaussianoUnCanal();
82          //filtros.EscribirImagen(imagen+"Base",operacion,formato);
83          filtros.DeteccionBordes();
84        }else if (operacion == "MB")
85        {
86          //(MB) Difuminado de movimiento (motion blur).
87          cout << "Motion" << endl;
88          filtros.DifuminadoMovimiento();
89        }else if (operacion == "SAP")
90        {
91          //(S&P) Ruido sal y pimienta
92          cout << "Sal y pimienta" << endl;
93          filtros.RuidoSalPimienta(0.5);
94        }else if (operacion == "E")
95        {
96          //(E) Erosi n
97          cout << "Erosion" << endl;
98          filtros.Binario();
99          //filtros.EscribirImagen(imagen+"Binaria",operacion,formato);
100         filtros.Erosion();
101       }else if (operacion == "D")
102       {
103         //(D) Dilataci n
104         cout << "Dilatacion" << endl;
105         filtros.Binario();
106         //filtros.EscribirImagen(imagen+"Binaria",operacion,formato);
107         filtros.Dilatacion();
108       }else if (operacion == "I")
109       {
110         //(I) Inversi n de color
111         cout << "Inversion" << endl;
112         filtros.InversionColor();
113       }else if (operacion == "G")
```

```
114         {
115             //(G) Transformaci n de escala de grises.
116             cout << "Grises" << endl;
117             filtros.TransformacionEscalaGrises();
118         }
119         //si se desea guardar como gif se guarda en bmp con
120         //opencv y se convierte a gif con imagemagick
121         if ((formato == "gif") || (formato == "GIF")){
122             //cout << "aqui"<< endl;
123             filtros.EscribirImagen(imagen,operacion,"bmp");
124             //cout << "o aqui"<< endl;
125             //codigo tomado de https://imagemagick.org/script/magick++.php
126             InitializeMagick(*argv);
127             // Construct the image object. Seperating image construction from the
128             // the read operation ensures that a failure to read the image file
129             // doesn't render the image object useless.
130             Image imagegif;
131             try {
132               // Read a file into image object
133               string ruta = "./"+imagen+"."+operacion+"."+"bmp";
134               string ruta2 = "./"+imagen+"."+operacion+"."+"gif";
135               imagegif.read(ruta);
136               // Crop the image to specified size (width, height, xOffset, yOffset)
137               //image.crop( Geometry(250,250));//;, 100, 100) );
138               // Write the image to a file
139               imagegif.write(ruta2);
140               string comando= "rm "+ruta;
141               int s = system(comando.c_str());
142               s=s;
143             }
144             catch( Magick::Exception &error_ )
145             {
146               cout << "Caught exception: " << error_.what() << endl;
147               return 1;
148             }
149         }else{
150             filtros.EscribirImagen(imagen,operacion,formato);
151         }
152     }else{
153         cout << "Revise los datos de entrada" << endl;
154     }
155     return 0;
156 }
```

Filtros.cpp:

```cpp
/**
 * @file Filtros.cpp
 */


#include "../include/Includes.hpp"

using namespace std;
using namespace cv;


/**
 * @brief Funcion que escribe una imagen
 * @param imagen es la imagen original
 * @param filtro es el filtro aplocado
 * @param formato es el formato de salida
 */
void Filtros::EscribirImagen(string imagen, string filtro, string formato){
    string ruta = "./"+imagen+"."+filtro+"."+formato;
    imwrite(ruta, matrizImagen);
}



/**
 * @brief Genera un filtro gausiano para imagenes de un canal
 */
void Filtros::FiltroGaussianoUnCanal(){
    Mat imagen = matrizImagen;
    Mat nueva(imagen.rows, imagen.cols, CV_8UC1);

    //FiltroGauss
    float fprom=0;
    //int ValorEstandar = 0;
    //int distancia=0;
    //int valor = 1;

    //se estudia el vecindario y se le asigna el promedio
    for (int i=1; i<imagen.rows-1; i++){
      for (int j=1; j<imagen.cols-1; j++){
        fprom = 0;
        for (int x = -1; x < 2; x++){
          for (int y = -1; y<2; y++){
            if ((x == 0) && (y == 0)){ //Este if evita el pixel central, haciendo
                                       //que el programa solo se centre en los del vecindario
            }else{
              fprom=fprom + imagen.at<uchar>(i+x, j+y);
            }
          }
        }
        fprom = fprom/8;
        nueva.at<uchar>(i, j) = fprom;
      }
    }

    //imshow("nueva", nueva);
    //std::cout<<"Valor azul: "<<pixel[0]<<endl;
```

```
58      //std::cout<<"Valor verde: " <<pixel[1]<<endl;
59      //std::cout<<"Valor verde: " <<pixel[2]<<endl;
60      //waitKey(0);
61      matrizImagen = nueva;
62  }
63
64
65  /**
66   * @brief Genera un filtro gausiano
67   */
68  void Filtros::FiltroGaussiano(){
69      Mat imagen = matrizImagen;
70      Mat nueva(imagen.rows, imagen.cols, CV_8UC3);
71
72      //FiltroGauss
73      int TriPromediosCanal[3]; //Almacena,primero, la suma de cada valor de pixel del
74      //vecindario para cada canal, en las posiciones BGR; luego, los promedios de estos
75      //valores
76      Vec3b tripleta1;//Almacena, temporalmente, los valores de pixel de cada canal
77      Vec3b pixel;
78      Vec3b TriValoresPixel;
79      float fprom=0;
80      int iprom;
81      //int ValorEstandar = 0;
82      //int distancia=0;
83      //int valor = 1;
84
85      //se estudia el vecindario y se le asigna el promedio
86      for (int i=1; i<imagen.rows-1; i++){
87        for (int j=1; j<imagen.cols-1; j++){
88            for (int colorp = 0; colorp < 3; colorp++){
89                fprom = 0;
90                for (int x = -1; x < 2; x++){
91                  for (int rep = -1; rep<2; rep++){
92                      if (x == 0 and rep == 0){ //Este if evita el pixel central,
93                                                //haciendo que el programa solo se
94                                                //centre en los del vecindario
95                      }else{
96                        tripleta1=imagen.at<Vec3b>(i+x, j+rep);
97                        fprom += tripleta1[colorp];
98                      //Recibe un color de tripleta 1 basandose
99                      //en el numero de repeticion colorp
100
101                     }
102                 }
103             }
104             iprom = int(fprom/8);
105             TriPromediosCanal[colorp]=iprom;
106                     pixel[colorp]=TriPromediosCanal[colorp];
107         }
108       nueva.at<Vec3b>(i, j)=pixel;
109       }
110     }
111
112     //imshow("nueva", nueva);
113     //std::cout<<"Valor azul: "<<pixel[0]<<endl;
114     //std::cout<<"Valor verde: " <<pixel[1]<<endl;
115     //std::cout<<"Valor verde: " <<pixel[2]<<endl;
116     //waitKey(0);
```

```
117     matrizImagen = nueva;
118 }
119
120 /**
121  * @brief Funcion que realiza el filtro de desviavion estandar
122  */
123 void Filtros::FiltroDesviacionEstandar(){
124     Mat imagen = matrizImagen;
125     Mat nueva(imagen.rows, imagen.cols, CV_8UC3);
126
127     //FiltroGaussiano
128     int TriPromediosCanal[3]; //Almacena,primero, la suma de cada valor de pixel del
129                     //vecindario para cada canal, en las posiciones BGR; luego,
130                     //los promedios de estos valores
131     Vec3b tripleta1;//Almacena, temporalmente, los valores de pixel de cada canal
132     Vec3b pixel;
133     Vec3b TriValoresPixel;
134     float fprom=0;
135     int iprom;
136     //int ValorEstandar = 0;
137     int distancia=0;
138     //int valor = 1;
139
140
141     //cacula la desviacion estandar en el vecindario y la asigna al pixel
142     for (int i=1; i<imagen.rows-1; i++){
143       for (int j=2; j<imagen.cols-1; j++){
144
145           for (int colorp = 0; colorp < 3; colorp++){
146               fprom = 0;
147               distancia = 0;
148               for (int x = -1; x < 2; x++){
149                 for (int rep = -1; rep<2; rep++){
150                     if (x == 0 and rep == 0){ //Este if evita el pixel central,
151                                               //haciendo que el programa solo se
152                                               //centre en los del vecindario
153                     }else{
154                       tripleta1=imagen.at<Vec3b>(i+x, j+rep);
155                       fprom += tripleta1[colorp];//Recibe un color de tripleta 1
156                                 //basandose en el numero de repeticion colorp
157
158                     }
159                 }
160               }
161               iprom = int(fprom/8);
162               TriPromediosCanal[colorp]=iprom;
163               for (int x = -1; x < 2; x++){
164                 for (int rep = -1; rep<2; rep++){
165                     if (x == 0 and rep == 0){ //Este if evita el pixel central,
166                                               //haciendo que el programa solo se centre
167                                               //en los del vecindario
168                     }else{
169                       tripleta1=imagen.at<Vec3b>(i+x, j+rep);
170                       distancia+=pow(tripleta1[colorp]-TriPromediosCanal[colorp],2);
171
172                     }
173                 }
174               }
175
```

```
176                                 //int raiz =
177                         pixel[colorp]= sqrt(distancia/3);
178
179
180             }
181
182             nueva.at<Vec3b>(i, j)=pixel;
183         }
184     }
185
186     //imshow("nueva", nueva);
187     //waitKey(0);
188     matrizImagen=nueva;
189 }
190
191 /**
192  * @brief Genera un filtro negativo
193  */
194 void Filtros::InversionColor(){
195     Mat imagen = matrizImagen;
196     Mat nueva(imagen.rows, imagen.cols, CV_8UC3);
197     Vec3b pixel;
198     int PixelTemporal;
199     for (int i=0; i<imagen.rows; i++){
200       for (int j=0; j<imagen.cols; j++){
201     pixel=imagen.at<Vec3b>(i, j);
202     for (int k=0; k<3; k++){
203       PixelTemporal=pixel[k];
204       pixel[k]=255-PixelTemporal;
205     }
206     nueva.at<Vec3b>(i, j)=pixel;
207   }
208     }
209     //cout<<imagen.channels();
210     //imshow("nueva", nueva);
211     //waitKey(0);
212     matrizImagen=nueva;
213 }
214
215 /**
216  * @brief Genera un filtro en escala de grises
217  */
218 void Filtros::TransformacionEscalaGrises(){
219     Mat imagen = matrizImagen;
220     Mat nueva(imagen.rows, imagen.cols, CV_8UC1);
221     //imshow("Original", imagen);
222     //FiltroBNegro
223     for (int i=0; i<imagen.rows; i++){
224       for (int j=0; j<imagen.cols; j++){
225     Vec3b pixel = imagen.at<Vec3b>(i, j);
226     int B = pixel[0];
227     int G = pixel[1];
228     int R = pixel[2];
229     int ValorTriMedio = (B + G + R)/3;
230         pixel[0] = ValorTriMedio;
231         pixel[1] = ValorTriMedio;
232         pixel[2] = ValorTriMedio;
233     //nueva.at<Vec3b>(i, j)=pixel;
234         nueva.at<uchar>(i, j)=ValorTriMedio;
```

```
235         }
236     }
237     //imshow("nueva", nueva);
238     //waitKey(0);
239     matrizImagen = nueva;
240 }
241
242 /**
243  * @brief Filtro de movimiento
244  */
245 void Filtros::DifuminadoMovimiento(){
246     Mat imagen = matrizImagen;
247     Mat nueva(imagen.rows, imagen.cols, CV_8UC3);
248     //imshow("Original", imagen);
249
250
251     //FiltroMovimiento
252     Vec3b pixel;
253     Vec3b tripleta1;
254     Vec3b tripleta2;
255     Vec3b tripleta3;
256     int promedio_rojo;
257     int promedio_verde;
258     int promedio_azul;
259     for (int k=0; k<4; k++){//Aqui se puede variar el valor de 4, aumentandolo para
260                             //difuminar mas la imagen
261       for (int i=0; i<imagen.rows; i++){
262     for (int j=0; j<imagen.cols; j++){
263       pixel=imagen.at<Vec3b>(i, j);
264       if (j!=0 && j!=imagen.cols){
265         tripleta1=imagen.at<Vec3b>(i, j-1);
266         tripleta2=imagen.at<Vec3b>(i, j);
267         tripleta3=imagen.at<Vec3b>(i, j+1);
268         promedio_azul=((tripleta1[0])+(tripleta2[0])+(tripleta3[0]))/3;
269         promedio_verde=((tripleta1[1])+(tripleta2[1])+(tripleta3[1]))/3;
270         promedio_rojo=((tripleta1[2])+(tripleta2[2])+(tripleta3[2]))/3;
271         pixel[0]=promedio_azul;
272         pixel[1]=promedio_verde;
273         pixel[2]=promedio_rojo;
274       }
275       nueva.at<Vec3b>(i, j)=pixel;
276
277     }
278   }
279   imagen=nueva;
280     }
281     //cout<<imagen.channels();
282     //imshow("nueva", nueva);
283     //waitKey(0);
284     matrizImagen = nueva;
285 }
286
287 /**
288  * @brief genera un numero aleatorio con distribucion uniforme
289  * @param tope es el numero mayor que se va a generar
290  */
291 int numeroRandom(int tope){
292     //codigo tomado de https://es.stackoverflow.com/questions/148661/por-qu%
293     //C3%A9-el-n%C3%BAmero-que-me-genera-el-rand-siempre-es-el-mismo
```

```
294     // Tenemos control sobre el algoritmo y distribuci n a usar.
295     random_device device;
296     // Se usa el algoritmo Mersenne twister
297     // https://es.wikipedia.org/wiki/Mersenne_twister
298     mt19937 generador(device());
299     // Escogemos una distribucion uniforme entre 0 y 100
300     uniform_int_distribution<> distribucion(0, tope);
301     /* Generamos un n mero pseudo-aleatorio con el algoritmo
302     mt19937 distribuido uniformemente entre 0 y 100 */
303     int a = distribucion(generador);
304     return a;
305 }
306
307
308 /**
309  * @brief Creal el filtro sal y pimienta
310  * @param porciento es el porcentaje de aleatoriedad del filtro
311  */
312 void Filtros::RuidoSalPimienta(float porciento){
313     Mat imagen = matrizImagen;
314     Vec3b pixel;
315     Vec3b blanco = {255,255,255};
316     Vec3b negro = {0,0,0};
317     float porcentaje = porciento;
318     float pTotales = imagen.rows * imagen.cols;
319     float pFrecuencia = pTotales * (porcentaje/10000);
320     int contador = 0;
321     int frecuencia = round(pFrecuencia);
322     for (int i=0; i<imagen.rows; i++){
323       for (int j=0; j<imagen.cols; j++){
324             contador++;
325         if ((contador %frecuencia)== 0){
326                 int nRandom = numeroRandom(5);
327                 if (nRandom == 0){
328                     imagen.at<Vec3b>(i,j) = blanco;
329                 }else if(nRandom == 1){
330                     imagen.at<Vec3b>(i,j) = negro;
331                 }
332             }
333       }
334     }
335     //cout<<imagen.channels();
336     //imshow("nueva", nueva);
337     //waitKey(0);
338     matrizImagen=imagen;
339 }
340
341
342
343 /**
344  * @brief Crea el filtro de imagen binaria
345  */
346 void Filtros::Binario(){
347     Mat imagen = matrizImagen;
348     Mat nueva(imagen.rows, imagen.cols, CV_8UC3);
349     Vec3b pixel;
350     Vec3b blanco = {255,255,255};
351     Vec3b negro = {0,0,0};
352     for (int i=0; i<imagen.rows; i++){
```

```
353       for (int j=0; j<imagen.cols; j++){
354     pixel=imagen.at<Vec3b>(i, j);
355         double gris = pixel[0]*0.3 + pixel[1]*0.59+pixel[2]*0.11;
356             if (gris >127){
357                 nueva.at<Vec3b>(i, j)=blanco;
358             }else{
359                 nueva.at<Vec3b>(i, j)=negro;
360             }
361       }
362     }
363     matrizImagen = nueva;
364 }
365
366
367
368 /**
369  * @brief Crea el filtro de erosion
370  */
371 void Filtros::Erosion(){
372     Mat imagen = matrizImagen;
373     Mat nueva(imagen.rows, imagen.cols, CV_8UC3);
374     Vec3b pixel;
375     Vec3b blanco = {255,255,255};
376     Vec3b negro = {0,0,0};
377     /*for (int i=0; i<imagen.rows; i++){
378       for (int j=0; j<imagen.cols; j++){
379     pixel=imagen.at<Vec3b>(i, j);
380         double gris = pixel[0]*0.3 + pixel[1]*0.59+pixel[2]*0.11;
381             if (gris >127){
382                 nueva.at<Vec3b>(i, j)=blanco;
383             }else{
384                 nueva.at<Vec3b>(i, j)=negro;
385             }
386       }
387     }*/
388
389     //elemento estructurante
390     int tamanio = 3;
391     Vec3b estructurante[tamanio][tamanio];
392     for (int i = 0;i<tamanio;i++){
393         for(int j=0;j<tamanio;j++){
394             estructurante[i][j] = blanco;
395         }
396     }
397     //recorremos la imagen binaria
398     //se estudia el vecindario y se cumple con el elemento estructurante
399     //se cambia el pixel
400     for (int i=1; i<imagen.rows-1; i++){
401       for (int j=1; j<imagen.cols-1; j++){
402             Vec3b vecindario[tamanio][tamanio];
403             int cof=0;
404             for (int fv = (i-1);fv<(i-1)+tamanio;fv++){
405                 int coc=0;
406                 for(int cv=(j-1);cv<(j-1)+tamanio;cv++){
407                     vecindario[cof][coc] = imagen.at<Vec3b>(fv, cv);
408                     coc++;
409                 }
410                 cof++;
411             }
```

```
412             int pasa = 1;
413             for (int f=0;f < tamanio;f++){
414                 for (int c = 0; c < tamanio;c++){
415                     if (estructurante[f][c] != vecindario[f][c]){
416                         pasa=0;
417                     }
418                 }
419             }
420             if (pasa == 0){
421                 nueva.at<Vec3b>(i, j) = negro;
422             }else{
423                 nueva.at<Vec3b>(i, j) = blanco;
424             }
425       }
426     }
427
428
429     //cout<<imagen.channels();
430     //imshow("nueva", nueva);
431     //waitKey(0);
432     matrizImagen=nueva;
433
434 }
435
436
437 /**
438  * brief Crea el filtro de dilatacion
439  *
440 void Filtros::Dilatacion(){
441     Mat imagen = matrizImagen;
442     Mat nueva(imagen.rows, imagen.cols, CV_8UC3);
443     Vec3b pixel;
444     Vec3b blanco = {255,255,255};
445     Vec3b negro = {0,0,0};
446     for (int i=0; i<imagen.rows; i++){
447       for (int j=0; j<imagen.cols; j++){
448     pixel=imagen.at<Vec3b>(i, j);
449         double gris = pixel[0]*0.3 + pixel[1]*0.59+pixel[2]*0.11;
450             if (gris >127){
451                 nueva.at<Vec3b>(i, j)=blanco;
452             }else{
453                 nueva.at<Vec3b>(i, j)=negro;
454             }
455       }
456     }*/
457
458     //elemento estructurante
459     /*int tamanio = 3;
460     Vec3b estructurante[tamanio][tamanio];
461     for (int i = 0;i<tamanio;i++){
462         for(int j=0;j<tamanio;j++){
463             estructurante[i][j] = negro;
464         }
465     }*//*
466     int promedio  = 0;
467     Vec3b pixelActual;
468     //recorremos la imagen binaria
469     for (int i=1; i<imagen.rows-1; i++){
470       for (int j=1; j<imagen.cols-1; j++){
```

```
471             pixelActual = imagen.at<Vec3b>(i,j);
472             promedio = (imagen.at<Vec3b>(i, j)[0]
473             //elemento estructurante
474             int tamanio = 3;
475             Vec3b estructurante[tamanio];
476             for (int i = 0;i<tamanio;i++){
477                 for(int j=0;j<tamanio;j++){
478                     estructurante[i][j] = promedio;
479                     cout << (int)estructurante[i][j] << "  ";
480                 }
481                 cout << endl;
482             }
483             Vec3b vecindario[tamanio];
484             cout << endl;
485             cout << endl;
486             cout << endl;
487             int cof=0;
488             for (int fv = (i-1);fv<(i-1)+tamanio;fv++){
489                 int coc=0;
490                 for(int cv=(j-1);cv<(j-1)+tamanio;cv++){
491                     vecindario[cof][coc] = (imagen.at<V
492                     coc++;
493 
494                 }
495                 cof++;
496                 cout << endl;
497 
498             }
499 
500             cout << endl;
501             cout << endl;
502             cout << endl;
503             if ((i>=(imagen.rows)/2) && (j>=(imagen.cols/2))){
504             //    system("sleep 4");
505             }
506             int pasa = 1;
507             for (int f=0;f < tamanio;f++){
508                 for (int c = 0; c < tamanio;c++){
509                     if (estructurante[f][c] != vecindario[f][c]){
510                         pasa=0;
511                     }
512                 }
513             }
514             if (pasa == 0){
515                 nueva.at<Vec3b>(i, j) = pixelActual;
516             }else{
517                 promedio = 0;
518                 for (int i = 0; i< tamanio ; i++){
519                     for (int j = 0; j < tamanio; j++){
520                         promedio = promedio + vecindario[i][j];
521                     }
522                 }
523                 promedio = promedio /9;
524                 pixelActual[0] = promedio;
525                 pixelActual[1] = promedio;
526                 pixelActual[2] = promedio;
527                 nueva.at<Vec3b>(i, j) = pixelActual;
528             }
529         }
```

```
530     }
531
532
533     //cout<<imagen.channels();
534     //imshow("nueva", nueva);
535     //waitKey(0);
536     matrizImagen=nueva;
537
538 }
539 */
540
541
542
543
544
545
546 /**
547  * @brief Crea el filtro de dilatacion
548  */
549 void Filtros::Dilatacion(){
550     Mat imagen = matrizImagen;
551     Mat nueva(imagen.rows, imagen.cols, CV_8UC3);
552     Vec3b pixel;
553     Vec3b blanco = {255,255,255};
554     Vec3b negro = {0,0,0};
555     /*for (int i=0; i<imagen.rows; i++){
556       for (int j=0; j<imagen.cols; j++){
557     pixel=imagen.at<Vec3b>(i, j);
558         double gris = pixel[0]*0.3 + pixel[1]*0.59+pixel[2]*0.11;
559             if (gris >127){
560                 nueva.at<Vec3b>(i, j)=blanco;
561             }else{
562                 nueva.at<Vec3b>(i, j)=negro;
563             }
564       }
565     }*/
566
567     //elemento estructurante
568     int tamanio = 3;
569     Vec3b estructurante[tamanio][tamanio];
570     for (int i = 0;i<tamanio;i++){
571         for(int j=0;j<tamanio;j++){
572             estructurante[i][j] = negro;
573         }
574     }
575     //recorremos la imagen binaria
576     //se estudia el vecindario y se cumple con el elemento estructurante
577     //se cambia el pixel
578     for (int i=1; i<imagen.rows-1; i++){
579       for (int j=1; j<imagen.cols-1; j++){
580             Vec3b vecindario[tamanio][tamanio];
581             int cof=0;
582             for (int fv = (i-1);fv<(i-1)+tamanio;fv++){
583                 int coc=0;
584                 for(int cv=(j-1);cv<(j-1)+tamanio;cv++){
585                     vecindario[cof][coc] = pixel=imagen.at<Vec3b>(fv, cv);;
586                     coc++;
587                 }
588                 cof++;
```

```
589             }
590             int pasa = 1;
591             for (int f=0;f < tamanio;f++){
592                 for (int c = 0; c < tamanio;c++){
593                     if (estructurante[f][c] != vecindario[f][c]){
594                         pasa=0;
595                     }
596                 }
597             }
598             if (pasa == 0){
599                 nueva.at<Vec3b>(i, j) = blanco;
600             }else{
601                 nueva.at<Vec3b>(i, j) = negro;
602             }
603       }
604     }
605 
606 
607     //cout<<imagen.channels();
608     //imshow("nueva", nueva);
609     //waitKey(0);
610     matrizImagen=nueva;
611 
612 }
613 
614 /**
615  * @brief calcula los bordes de una imagen
616  */
617 void Filtros::DeteccionBordes(){
618     Mat imagen = matrizImagen;
619     Mat direccionX(imagen.rows, imagen.cols, CV_8UC1);
620     Mat direccionY(imagen.rows, imagen.cols, CV_8UC1);
621     Mat resultado(imagen.rows, imagen.cols, CV_8UC1);
622     Mat temporal(imagen.rows, imagen.cols, CV_8UC1);
623     float angulos[imagen.rows][imagen.cols];
624     int Gx[3][3] = {{-1, 0 ,1},
625                     {-2, 0 ,2},
626                     {-1, 0 ,1 }};
627     int Gy[3][3] = {{-1, -2 ,-1},
628                     { 0,  0 , 0},
629                     { 1,  2 , 1}};
630     //calculamos el gradiente
631     int pixel;
632     for (int i=1; i<imagen.rows-1; i++){
633       for (int j=1; j<imagen.cols-1; j++){
634             int calculoX = 0;
635             int calculoY = 0;
636         for (int x = -1; x < 2; x++){
637           for (int y = -1; y<2; y++){
638                     pixel=imagen.at<uchar>(i+x, j+y);
639             if ((x == 0) && (y == 0)){
640             }else{
641               calculoX = calculoX + (Gx[x+1][y+1] * pixel);
642                         calculoY = calculoY + (Gy[x+1][y+1] * pixel);
643             }
644           }
645         }
646             direccionX.at<uchar>(i, j) = calculoX/8;
647             direccionY.at<uchar>(i, j) = calculoY/8;
```

```
    }
  }
  //calculamos la raiz cuadrada de las potencias
  int raiz = 0;
  for (int i=1; i<imagen.rows-1; i++){
    for (int j=1; j<imagen.cols-1; j++){
        raiz = sqrt( pow(direccionX.at<uchar>(i, j),2)+ pow(direccionY.at<uchar>(i, j),2));
        resultado.at<uchar>(i, j) = raiz;
    }
  }

  //calculamos el angulo de la gradiente
  double angulo = 0;
  for (int i=1; i<imagen.rows-1; i++){
    for (int j=1; j<imagen.cols-1; j++){
        int x = direccionX.at<uchar>(i, j);
        int y = direccionY.at<uchar>(i, j);
        if (x != 0){
            angulo = (atan ( y/ x)  * 180 )/ M_PI;
            if ((angulo >= 0) && (angulo <=22)){
                angulo = 0;
            }else if ((angulo > 22) && (angulo <=67)){
                angulo = 45;
            }
            else if ((angulo > 67) && (angulo <=112)){
                angulo = 90;
            }else if ((angulo > 112) && (angulo <=157)){
                angulo = 135;
            }else if ((angulo > 127) && (angulo <=180)){
                angulo = 0;
            }
        }
        angulos[i][j] = angulo;
    }
  }

  //estudiamos el vecinadario en direccion al angulo
  int pixelCentral;
  int pixelVecino;
  int direccion0[3][3]={{0,0,0},{1,1,1},{0,0,0}};
  int direccion45[3][3]={{0,0,1},{0,1,0},{1,0,0}};
  int direccion90[3][3]={{0,1,0},{0,1,0},{0,1,0}};
  int direccion135[3][3]={{1,0,0},{0,1,0},{0,0,1}};
  for (int i=1; i<resultado.rows-1; i++){
    for (int j=1; j<resultado.cols-1; j++){
        pixelCentral = resultado.at<uchar>(i, j);
        int pasa = 1;
      for (int x = -1; x < 2; x++){
        for (int y = -1; y<2; y++){
            if (angulos[i][j] == 90){
                pixelVecino=resultado.at<uchar>(i+x, j+y)*direccion0[x+1][y+1];
            }else if (angulos[i][j] == 135){
                pixelVecino=resultado.at<uchar>(i+x, j+y)*direccion45[x+1][y+1];
            }else if (angulos[i][j] == 0){
                pixelVecino=resultado.at<uchar>(i+x, j+y)*direccion90[x+1][y+1];
            }else if (angulos[i][j] == 45){
                pixelVecino=resultado.at<uchar>(i+x, j+y)*direccion135[x+1][y+1];
            }else{
```

```
                        pixelVecino = 0;
                    }
                    if (pixelVecino > pixelCentral){
                        pasa = 0;
                    }
            }
        }
            if (pasa == 0){
                temporal.at<uchar>(i, j) = 0;
            }else{
                temporal.at<uchar>(i, j) = pixelCentral;
            }
      }
    }

    resultado = temporal;
    //llenamos el borde de negro
    for (int i=0;i<resultado.rows; i++){
        for (int j=0;j<resultado.cols; j++){
            if (i < 2){
                resultado.at<uchar>(i, j) = 0;
            }
            if (j < 2){
                resultado.at<uchar>(i, j) = 0;
            }
            if (i > (resultado.rows -2)){
                resultado.at<uchar>(i, j) = 0;
            }
            if (j > (resultado.cols -2)){
                resultado.at<uchar>(i, j) = 0;
            }
        }
    }
    matrizImagen = resultado;

    //imshow("nueva", nueva);
    //std::cout<<"Valor azul: "<<pixel[0]<<endl;
    //std::cout<<"Valor verde: " <<pixel[1]<<endl;
    //std::cout<<"Valor verde: " <<pixel[2]<<endl;
    //waitKey(0);


}
```

Filtros.hpp:

```
/**
 *@file Filtros.hpp/
 */
#ifndef FILTROS_HPP
#define FILTROS_HPP

#include "./Includes.hpp"

using namespace std;
using namespace cv;

/**
 * @brief Clase que controla todos los filtros
 */
class Filtros
{
    public:
        /**
        * @brief constructor
        */
        Filtros(string imagen){
            Mat abrirImagen = imread(imagen,CV_LOAD_IMAGE_COLOR);
            matrizImagen = abrirImagen;
        }
        /**
        * @brief constructor por defecto
        */
        Filtros(){
        }
        /**
        * @brief destructor
        */
        ~Filtros(){
        };
        //funciones de la clase
        void FiltroGaussiano();
        void FiltroGaussianoUnCanal();
        void FiltroDesviacionEstandar();
        void DeteccionBordes();
        void DifuminadoMovimiento();
        void RuidoSalPimienta(float);
        void Erosion();
        void Dilatacion();
        void InversionColor();
        void TransformacionEscalaGrises();
        void Binario();
        void EscribirImagen(string, string, string);
        Mat matrizImagen;


/*
(FG) Filtro gaussiano.
(FSTD) Filtro de desviaci n est ndar.
(ED) Detecci n de bordes (edge detection).
(MB) Difuminado de movimiento (motion blur).
(S&P) Ruido sal y pimienta
(E) Erosi n
```

```
58 (D) Diltaci n
59 (I) Inversi n de color
60 (G) Transformaci n de escala de grises.
61 */
62 };
63
64 #endif
```

Includes.hpp:

```
/**
 *@file Includes.hpp/
 */
#ifndef INCLUDES_H
#define INCLUDES_H
    /*#include <stdio.h>
    #include <iostream>
    //#include <graphics.h>
    #include <opencv/cv.hpp>
    #include <opencv2/highgui.hpp>
    #include <math.h>
    #include <fstream>
    #include <random>
    #include <chrono>
    #include <stdlib.h>
    #include "./Filtros.hpp"
    //#include <opencv/cv.h>
    //#include <opencv/highgui.h>
    */
    #include <opencv2/highgui.hpp>
    #include <opencv/cv.hpp>
    #include<iostream>
    #include<stdlib.h>
    #include <chrono>
    #include <random>
    #include <fstream>
    #include <math.h>
    #include <cmath>
    #include "./Filtros.hpp"
    #include <random>
    #include <Magick++.h>

#endif
```