Transformación de Color

Alumno: Josue Samuel Philco Puma

18 de mayo de 2025

1. Función para la transformación de color

Se nos pide hacer la transformación de color a partir de una imagen, para ello debemos crear una función que lo haga, este recibe como parámetros la imagen original y una paleta de colores para que retorne la imagen original transformada con la paleta de colores indicada, esto se puede realizar con funciones predeterminadas de OpenCV como **cv::LUT()** y **cv::applyColorMap()**, pero nosotros crearemos nuestra propia función para lograrlo. Entonces, empezemos enumerando lo que vamos a realizar.

1. Siempre es recomendable que al cambiar una imagen debemos transformarla a escala grises, por lo que debemos almacenarlo en una nueva imagen.
2. Ahora, como vamos a retornar la imagen transformada debemos también almacenarla en otra nueva imagen.
3. Ahora, debemos cambiar nuestra imagen a escala grises, esto podemos hacer usando la función de OpenCV **cvtColor()**, pero también podemos hacerlo de forma manual accediendo a los píxeles y aplicar una pequeña fórmula para cambiarlo que es **0.299 * R** + **0.587 * G** + **0.114 * B**, una vez hecho se puede transformar la imagen en gris a otra imagen con el mapa de colores.
4. Ahora tambien si una imagen es gris, simplemente también se le aplica la transformación a gris, esto no afectará en nada a la imagen como tal, y luego se transforma la imagen gris a la imagen transformada con el mapa de colores.
5. Por último retornamos la imagen tranformada para poder mostrarla

Ahora con los pasos enumerados podemos hacer la función pedira para realizar la transformación de la imagen.

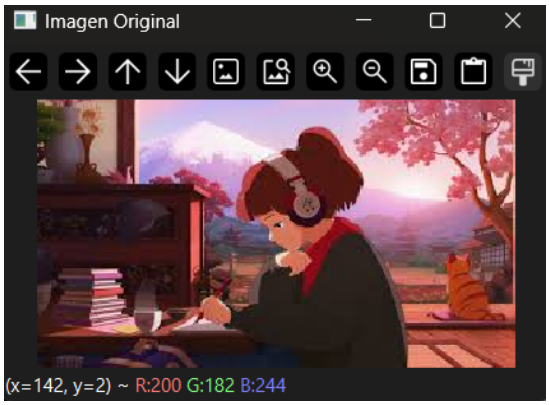
```
Mat create_image_personalized(const Mat& image, const vector<Vec3b>& color_map)
    {
    Mat image_gray(image.rows, image.cols, CV_8UC1);
    Mat image_transformed(image.rows, image.cols, CV_8UC3);

    if (image.channels() == 3) {
        for (int i = 0; i < image.rows; i++) {
            for (int j = 0; j < image.cols; j++) {
                Vec3b pixel = image.at<Vec3b>(i, j);
                uchar gray_value = static_cast<uchar>(0.299 * pixel[2] + 0.587
                    * pixel[1] + 0.114 * pixel[0]);
                image_gray.at<uchar>(i, j) = gray_value;

                Vec3b color_value = color_map[gray_value];
                image_transformed.at<Vec3b>(i, j) = color_value;
            }
        }
        imshow("Imagen en Escala de Grises", image_gray);
    }
    else {
        image_gray = image.clone();
        for (int i = 0; i < image.rows; i++) {
            for (int j = 0; j < image.cols; j++) {
                uchar gray_value = image.at<uchar>(i, j);
                Vec3b color_value = color_map[gray_value];
                image_transformed.at<Vec3b>(i, j) = color_value;
            }
        }
    }
```

```
28      return image_transformed;
29  }
```

Como se muestra en el código, se cumple con los pasos anteriores, recibir la imagen con el mapa de colores para aplicar el mapa de colores en la imagen, la imagen la convertimos en gris y luego aplicar el mapa de colores para transformar la imagen, y luego retorna la imagen transformada con el mapa de colores. Como tal no podemos todavía ver la imagen transformada, ya que no contamos con la paleta de colores, pero podemos hacer una demostración de la conversión de la imagen a gris.



(a) Imagen original.

(b) Imagen original transformada a gris.

Figura 1: Mostrando la imagen original (a) y la imagen en gris (b)

2. Función debe trabajar con imágenes a color como en escala gris

Para esta parte nos pide que la función implementada también debe aceptar imágenes a color e imágenes en escala gris, esto sobretodo se refiere a que la imagen original lo pasamos a tranformar a escala gris no debería afectar en nada a la imagen original, es decir, la imagen gris no debe tener ni una modificación al transformarla, solo debería tener de resultado la imagen transformada con el mapa de colores. Entonces para realizar esto simplemente seguimos los mismos pasos del ejercicio anterior. En este caso se mostrará una imagen que está en gris y ver si no la transforma y la mantiene igual.

(a) Imagen original.

(b) Imagen transformada sin alterar.

Figura 2: Mostrando la imagen original en gris (a) y la misma imagen sin alteración(b)

Algo que aclarar es en los canales, en la primera imagen (a) se observa que tiene los 3 canales RGB, mientras que en la segunda imagen (b) ya no se muestra en el canal RGG, sino en un solo canal.

3. Establecer un mapa de color personalizado

Para este ejercicio debemos establecer un mapa de color perosnalizado, pero este mapa no debe estar incluido en un color map de OpenCV, si revisamos la documentación de color map habiendo un total de 22 mapas de colores que maneja OpenCV, pero en este caso nosotros mismos lo crearemos. Entonces empecemos a ver como lo realizaremos:

1. Primero debemos saber en donde alamcenarlo, para ello necesitariamos un vector de tamaño 256 para cada valor posible en la imagen.

2. Como buscamos también mostrar el mapa de colores deberiamos crear una imagen para que se logre visualizar este mapa.

3. Se recorre el rango de valores de gris desde 0 hasta 255, generando un color distinto para cada nivel. Para ello, se normaliza el índice usando la fórmula $t = \frac{i}{255,0}$, donde i es el valor de gris actual.

4. Con el valor normalizado podemos ir calculando los componentes de los 3 canales RGB y el color lo vamos almacenando en el vector creado y también va a pintar la imagen que creamos para el mapa de color.

5. Por último se debe mostrar el mapa de colores y a la vez retornar el vector con los valores almacenados, con eso tendríamos el mapa de color generado.

Aora que tenemos los pasos numerados veamos el código que hará este mapa de colores personalizado.

```
vector<Vec3b> color_map_personalized() {
    vector<Vec3b> color_map(256);
    Mat color_map_image(50, 256, CV_8UC3);
    for (int i = 0; i < 256; i++) {
        float t = i /255.0f;
        /* Primer Mapa de Color */
        // uchar b = static_cast<uchar>(255 * (1 - t));
        // uchar g = static_cast<uchar>(255*t);
        // uchar r = 128;

        /* Segundo Mapa de Color */
        uchar b = static_cast<uchar>(255 * (1 - t));
        uchar g = 64;
        uchar r = static_cast<uchar>(255*t);
        Vec3b c(b, g, r);
        color_map[i] = c;
        for (int j = 0; j < 50; j++) {
            color_map_image.at<Vec3b>(j, i) = c;
        }
    }
    imshow("Mapa de Color", color_map_image);
    return color_map;
% }
```

Ahora que tenemos esta función, podemos unirla a la otra función hecha anteriormente para visualizar nuestra imagen transformada con nuestro mapa de colores. A continuación se presenta los resultados:

(a) Imagen original.

(b) Mapa de color generado.

(c) Imagen transformada.

Figura 3: Imagen original (a), el mapa de colores (b) y la imagen transformada (c)

.

Para este primer resultado probamos un estilo de pasar de un canal Azul a un canal Verde manteniendo el canal Rojo en un número constante. Para mostrar otro ejemplo probaremos otra imagen y aplicar otro mapa distinto:

(a) Imagen original. (b) Mapa de color generado. (c) Imagen transformada.

Figura 4: Imagen original (a), el mapa de colores (b) y la imagen transformada (c)

.

En este caso pusimos en el canal Azul el mismo valor que el anterior, el canal Verde se mantiene en un valor constante y el color Rojo esta vez si aplica un cambio de color.

4. Transformación aplicada por separado a cada canal

Para este ejercicio debemos saber que pasaría si aplicamos esa transformación de imagen pero en cada canal RGB por separado y ver un resultado final. Para ello, debemos seguir una serie de pasos para lograrlo.

1. Primero podemos crear otra función que este dedicada a este funcionamiento y con los mismos parámetros que la anterior función.
2. Como estamos limitados de usar funciones predeterminadas de OpenCV como **cv::split** para dividir la imagen en los 3 canales, podemos reutilizar una técnica usada en el laboratorio 1 para separar las imágenes en cada canal.
3. Ahora podemos también crear una imágenes para ver como es que aplica la transformación de color en cada canal. Aunque no se pide explícitamente puede ser de gran valor.
4. Debemos recorrer la imagen para obtener el valor de los píxeles y aplicar el mapa de color a cada canal RGB.
5. Al final podríamos mostrar la combinación de cada canal para mostrar una sola imagen para ver si son coincidentes a la otra imagen transformada. Esto debemos lograrlo sacando un promedio entre los valores de los 3 canales.

Aunque esto llegue a funcionar, la pregunta sería: *¿Qué pasaría si lo hacemos?*. Bueno, veamos primero el código que realizará este funcionamiento:

```
1  void apply_color_map_channels(const Mat& image, const vector<Vec3b>& color_map)
       {
2      int rows = image.rows;
3      int cols = image.cols;
4
5      Mat color_map_channel_red(rows, cols, CV_8UC3);
6      Mat color_map_channel_green(rows, cols, CV_8UC3);
7      Mat color_map_channel_blue(rows, cols, CV_8UC3);
8      Mat imaged_transformed(rows, cols, CV_8UC3);
9
10     for (int i = 0; i < rows; i++) {
11         for (int j = 0; j < cols; j++) {
12             int channel_blue = image.at<Vec3b>(i, j)[0];
13             int channel_green = image.at<Vec3b>(i, j)[1];
14             int channel_red = image.at<Vec3b>(i, j)[2];
15
```

```
            Vec3b color_value_blue = color_map[channel_blue];
            Vec3b color_value_green = color_map[channel_green];
            Vec3b color_value_red = color_map[channel_red];

            color_map_channel_blue.at<Vec3b>(i, j) = color_value_blue;
            color_map_channel_green.at<Vec3b>(i, j) = color_value_green;
            color_map_channel_red.at<Vec3b>(i, j) = color_value_red;

            Vec3b color_avg;
            for (int k = 0; k < 3; k++) {
                color_avg[k] = static_cast<uchar>((color_value_blue[k] +
                    color_value_green[k] + color_value_red[k]) / 3);
            }
            imaged_transformed.at<Vec3b>(i, j) = color_avg;
        }
    }

    imshow("Canal Azul con Mapa de Color", color_map_channel_blue);
    imshow("Canal Verde con Mapa de Color", color_map_channel_green);
    imshow("Canal Rojo con Mapa de Color", color_map_channel_red);
    imshow("Imagen Transformada Unida", imaged_transformed);
}
```

Como vemos, se hace la separación de la imagen en los 3 canales para luego aplicar la transformación de color a cada canal por separado. Y al final se une esos 3 valores para generar una sola imagen para los 3 canales. Así que veamos los resultados que nos arroja:

(a) Canal Azul transformada.

(b) Canal Verde transformada.

(c) Canal Rojo transformada.

(d) Imagen transformada unida.

Figura 5: Mostrando la imagen transformada en sus canales separados y su unión al final

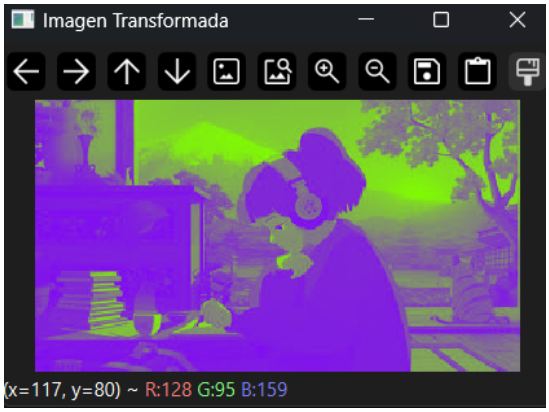
Entonces ahora que tenemos los resultados, se nos hace la pregunta: ***¿Qué ocurre si la transformación de color se aplica por separado a cada canal de una imagen RGB?***. Bueno, debemos primero saber si nuestra imagen unida al final coincide con la anterior imagen sin hacer la transformación en cada canal por separado. Así que veamos las dos imágenes.

(a) Imagen transformada sin separación.

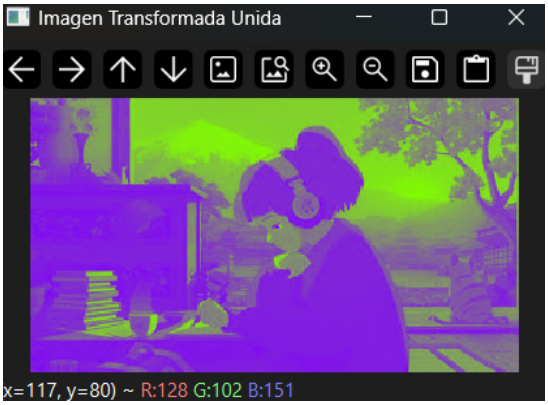
(b) Imagen transformada con separación.

Figura 6: Imagen sin separación de canales (a) e Imagen con separación de canales (b)

Viendo las imágenes se pueden ver muy similares a simple vista, pero que pasa si vemos sus valores píxel por píxel. Capaz si vemos los píxeles uno por uno haya diferentes valores en sus canales RGB. Por ejemplo veamos el píxel (117, 80) de cada imagen para ver si sus valores RGB son iguales.



(a) Viendo en el píxel (117, 80) de la imagen sin separación de canales.



(b) Viendo en el píxel (117, 80) de la imagen con separación de canales.

Figura 7: Viendo un píxel en común para ver su diferencia entre las dos imágenes

Bueno, revisamos el valor del píxel en el punto ***(117, 80)*** viendo valores diferentes en cada imagen. En la imagen sin separación de transformación en cada canal nos da un valor **RGB** de ***(128, 95, 159)***, mientras que, en la imagen que aplica la transformación en cada canal por separado nos da un **RGB** de ***(128, 102, 151)***. Entonces viendo ese punto podemos decir que ambas imágenes resultantes no son iguales, aunque a primera vista parezcan iguales los valores en los píxeles son muy diferentes. Para una mejor claridad sería mejor usar otro mapa de colores.

Bueno, también podríamos usar otro Mapa de Color para demostrarlo mejor, asi que podemos usar los siguientes valores en el mapa de color en la función que crea el mapa de color.

```
for (int i = 0; i < 256; i++) {
    float t = i /255.0f;
    uchar r = static_cast<uchar>(255 * t);
    uchar g = static_cast<uchar>(255 * (1 - t));
    uchar b = static_cast<uchar>(127 * sin(6.28f * t));

    Vec3b c(b, g, r);
    color_map[i] = c;
    for (int j = 0; j < 50; j++) {
        color_map_image.at<Vec3b>(j, i) = c;
    }
}
```

Con estos valores para el mapa de color podremos ver los diferentes. Así que veamos los resultados con este nuevo mapa.

(a) Imagen sin separación de canales con nuevo mapa de color.

(b) Imagen con separación de canales con nuevo mapa de color.

(c) Mapa de color utilizado para una mejor visualización.

Figura 8: Visualización más clara para ver los resultados

.

Entonces con estos resultados podemos dar la siguiente respuesta a este ejercicio: ***Cuando se aplica la transformación de color por separado a cada canal RGB, se obtiene un resultado visual diferente respecto a aplicar una única transformación sobre una imagen en escala de grises (como en la primera función create_image_personalized). Esto se debe a que***

cada canal (rojo, verde y azul) tiene su propia distribución de intensidades, por lo tanto, al usar el mismo mapa de color para cada uno de forma independiente, se generan tres imágenes transformadas distintas, una por canal, que luego se combinan. En el código apply_color_map_channels, a cada canal individual se le aplica el mismo color_map, y después los colores resultantes se promedian. Este promedio genera una imagen final donde el color resultante depende del valor individual de cada canal y cómo cada uno fue transformado por separado. Esto crea una transformación más rica y diversa en colores, pero menos coherente perceptualmente, ya que no respeta la luminancia real de la imagen.

5. Código completo de Laboratorio

Ahora se presenta todo el código completo con los pasos desarrollados:

```cpp
#include <bits/stdc++.h>
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;

vector<Vec3b> color_map_personalized() {
    vector<Vec3b> color_map(256);
    Mat color_map_image(50, 256, CV_8UC3);
    for (int i = 0; i < 256; i++) {
        float t = i /255.0f;
        /* Primer Mapa de Color */
        // uchar b = static_cast<uchar>(255 * (1 - t));
        // uchar g = static_cast<uchar>(255*t);
        // uchar r = 128;

        /* Segundo Mapa de Color */
        // uchar b = static_cast<uchar>(255 * (1 - t));
        // uchar g = 64;
        // uchar r = static_cast<uchar>(255*t);

        /* Tercer Mapa de Color */
        uchar r = static_cast<uchar>(255 * t);
        uchar g = static_cast<uchar>(255 * (1 - t));
        uchar b = static_cast<uchar>(127 * sin(6.28f * t));

        Vec3b c(b, g, r);
        color_map[i] = c;
        for (int j = 0; j < 50; j++) {
            color_map_image.at<Vec3b>(j, i) = c;
        }
    }
    imshow("Mapa de Color", color_map_image);
    return color_map;
}

Mat create_image_personalized(const Mat& image, const vector<Vec3b>& color_map)
    {
    Mat image_gray(image.rows, image.cols, CV_8UC1);
    Mat image_transformed(image.rows, image.cols, CV_8UC3);

    if (image.channels() == 3) {
        for (int i = 0; i < image.rows; i++) {
            for (int j = 0; j < image.cols; j++) {
                Vec3b pixel = image.at<Vec3b>(i, j);
                uchar gray_value = static_cast<uchar>(0.299 * pixel[2] + 0.587
                    * pixel[1] + 0.114 * pixel[0]);
                image_gray.at<uchar>(i, j) = gray_value;

                Vec3b color_value = color_map[gray_value];
```

```
                image_transformed.at<Vec3b>(i, j) = color_value;
            }
        }
        imshow("Imagen en Escala de Grises", image_gray);
    }
    else {
        image_gray = image.clone();
        for (int i = 0; i < image.rows; i++) {
            for (int j = 0; j < image.cols; j++) {
                uchar gray_value = image.at<uchar>(i, j);
                Vec3b color_value = color_map[gray_value];
                image_transformed.at<Vec3b>(i, j) = color_value;
            }
        }
    }
    return image_transformed;
}

void apply_color_map_channels(const Mat& image, const vector<Vec3b>& color_map)
    {
    int rows = image.rows;
    int cols = image.cols;

    Mat color_map_channel_red(rows, cols, CV_8UC3);
    Mat color_map_channel_green(rows, cols, CV_8UC3);
    Mat color_map_channel_blue(rows, cols, CV_8UC3);
    Mat imaged_transformed(rows, cols, CV_8UC3);

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int channel_blue = image.at<Vec3b>(i, j)[0];
            int channel_green = image.at<Vec3b>(i, j)[1];
            int channel_red = image.at<Vec3b>(i, j)[2];

            Vec3b color_value_blue = color_map[channel_blue];
            Vec3b color_value_green = color_map[channel_green];
            Vec3b color_value_red = color_map[channel_red];

            color_map_channel_blue.at<Vec3b>(i, j) = color_value_blue;
            color_map_channel_green.at<Vec3b>(i, j) = color_value_green;
            color_map_channel_red.at<Vec3b>(i, j) = color_value_red;

            Vec3b color_avg;
            for (int k = 0; k < 3; k++) {
                color_avg[k] = static_cast<uchar>((color_value_blue[k] +
                    color_value_green[k] + color_value_red[k]) / 3);
            }
            imaged_transformed.at<Vec3b>(i, j) = color_avg;
        }
    }

    imshow("Canal Azul con Mapa de Color", color_map_channel_blue);
    imshow("Canal Verde con Mapa de Color", color_map_channel_green);
    imshow("Canal Rojo con Mapa de Color", color_map_channel_red);
    imshow("Imagen Transformada Unida", imaged_transformed);
}

int main() {
    string image_file;
    cout << "Ingrese el nombre de la imagen (con extension): ";
    cin >> image_file;
    string image_path = "D:/UNSA EPCC/7mo semestre/Computacion Grafica/Unidad
        1/Imagenes con OpenCV/Imagenes/" + image_file;
```

```
    Mat image_original = imread(image_path);
    if (image_original.empty()) {
        cout << "No se pudo cargar la imagen." << endl;
        return -1;
    }
    vector<Vec3b> color_map = color_map_personalized();
    Mat image_transformed = create_image_personalized(image_original, color_map
        );
    apply_color_map_channels(image_original, color_map);
    imshow("Imagen Original", image_original);
    imshow("Imagen Transformada", image_transformed);
    waitKey(0);
    return 0;
}
```

Referencias

[1] OpenCV. *Color Maps - OpenCV documentation*. 2018. https://docs.opencv.org/3.4/d3/d50/group__imgproc__colormap.html. Accedido el 17 de mayo de 2025.

[2] International Telecommunication Union. *Recommendation ITU-R BT.601: Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios*. 2011. https://www.itu.int/rec/R-REC-BT.601/en. Accedido el 17 de mayo de 2025.

[3] TutorialsPoint. *OpenCV - Color Maps*. https://www.tutorialspoint.com/opencv/opencv_color_maps.htm. Accedido el 17 de mayo de 2025.

[4] Satya Mallick. *applyColorMap for Pseudo Coloring in OpenCV (C++/Python)*. 2018. https://learnopencv.com/applycolormap-for-pseudocoloring-in-opencv-c-python/. Accedido el 17 de mayo de 2025.