Instituto Politécnico Nacional
Escuela Superior de Cómputo

Práctica 3

Conversión entre modelos de color

Procesamiento Digital de Imágenes
Grupo 4BV1

Alumno: Padilla García Andrea Miranda

Objetivo: Implementar un código que haga las conversiones entre los modelos de color indicados

Introducción

Para esta práctica requerimos de conocer varios modelos de color, entre ellos están los siguientes:

- Modelo RGB (Red, Green, Blue): Representa el color como una combinación aditiva de los tres componentes primarios de la luz: rojo, verde y azul. Es el modelo estándar de visualización en monitores y cámaras digitales.
- Modelo CMY y CMYK (Cyan, Magenta, Yellow, Key/Black): Basado en la síntesis sustractiva de color, usado principalmente en impresión.
  CMY es el inverso de RGB, mientras que CMYK añade el componente K (negro) para mejorar la calidad de la impresión y reducir el gasto de tinta.
- Modelo YIQ: Utilizado en la televisión analógica NTSC. Separa la imagen en:
  - Y: luminancia (información de brillo, blanco y negro).
  - I y Q: crominancia (información de color).

  Esto permite transmitir imágenes a color sin perder compatibilidad con televisores en blanco y negro.
- Modelo HSI (Hue, Saturation, Intensity) Basado en la percepción humana del color.
  - H (Tono): color dominante (rojo, verde, azul, etc.).
  - S (Saturación): grado de pureza del color.
  - I (Intensidad): brillo percibido.

  Muy usado en segmentación y reconocimiento de objetos por ser más intuitivo que RGB.
- Modelo HSV (Hue, Saturation, Value): Similar al HSI, pero con V (Valor) en lugar de intensidad, definido como el componente más brillante entre R, G y B. Es útil en aplicaciones de edición gráfica e interfaces de usuario.
- Modelo CIE Lαβ (L, α, β): Modelo perceptualmente uniforme, diseñado para aproximarse a la visión humana.
  - L: luminosidad.
  - α: escala entre verde y rojo.
  - β: escala entre azul y amarillo.

  Muy utilizado en aplicaciones de comparación y transferencia de color, ya que separa de forma más natural la información de brillo y cromaticidad.

Adicional a las conversiones entre modelos de color se debe hacer una **transferencia de color,** una transferencia de color es una técnica de procesamiento de imágenes que permite modificar la paleta cromática de una imagen objetivo tomando como referencia otra imagen. El procedimiento común se basa en el modelo *CIE Lab*, debido a que sus componentes son más consistentes con la percepción humana del color.

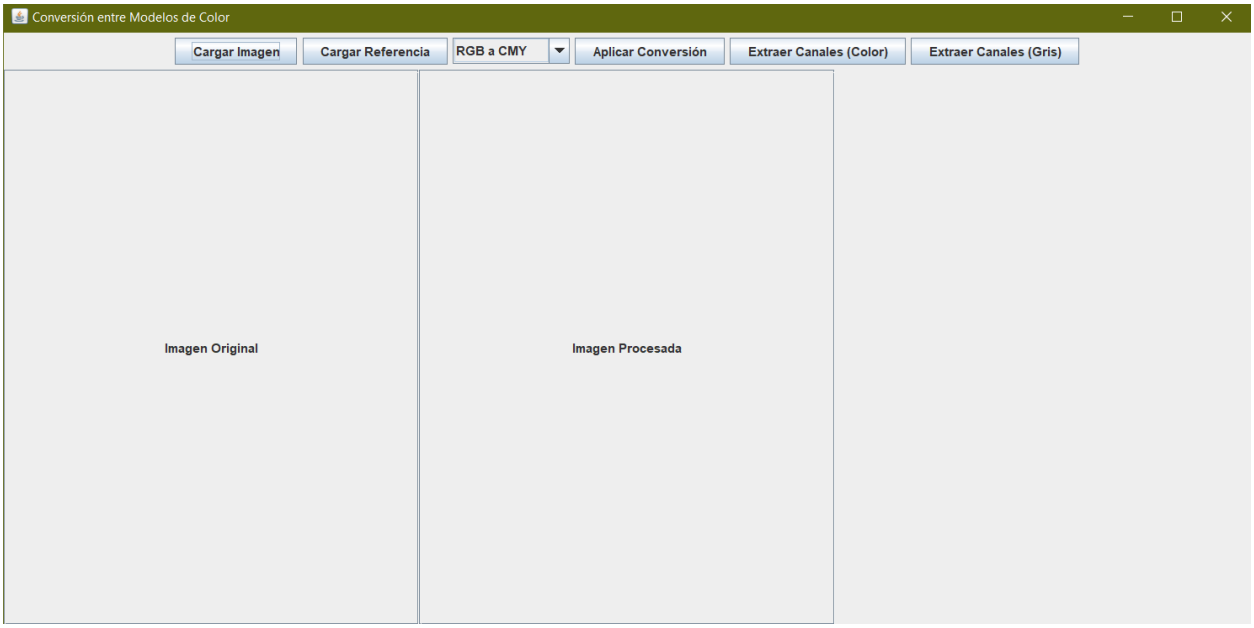
El proceso general consiste en:

1. Convertir ambas imágenes (original y referencia) al espacio Lab.
2. Calcular la media y desviación estándar de cada canal en la imagen de referencia.
3. Ajustar los valores de la imagen original, normalizando con respecto a su propia media y desviación estándar, y luego escalando con los parámetros de la imagen de referencia.
4. Convertir la imagen resultante nuevamente al espacio RGB para su visualización.

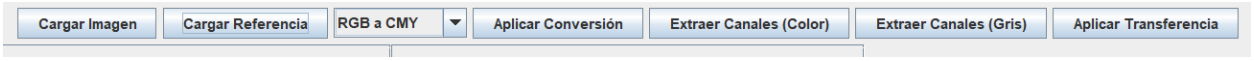
Este método permite aplicar el estilo cromático de una imagen a otra, conservando la estructura y detalles de la imagen original, pero adaptando su "atmósfera" de color.

Desarrollo

El diseño para la interfaz gráfica es el siguiente:



Al cargar la imagen de referencia se hacer visible el botón de "Aplicar Transferencia" para transferir los colores de la imagen de referencia a la original y mostrarla en el espacio "Imagen Procesada":



Los métodos de conversión se programan dentro de la lógica del programa

```java
//RGB --> CMY
public BufferedImage convertirRGBaCMY() {
    int w = imagen Original.getWidth();
    int h = imagen Original.getHeight();
    BufferedImage cmy = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagen Original.getRGB(x, y));
            int r = c.getRed();
            int g = c.getGreen();
            int b = c.getBlue();

            int C = 255 - r;
            int M = 255 - g;
            int Y = 255 - b;

            cmy.setRGB(x, y, new Color(C, M, Y).getRGB());
        }
    }
    this.image Procesada = cmy;
    return cmy;
}
```

```java
//CMY --> RGB
public BufferedImage convertirCMYaRGB() {
    int w = imagenProcesada.getWidth();
    int h = imagen Procesada.getHeight();
    BufferedImage rgb = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagen Procesada.getRGB(x, y));
            int C = c.getRed();
            int M = c.getGreen();
            int Y = c.getBlue();

            int R = 255 - C;
            int G = 255 - M;
            int B = 255 - Y;

            rgb.setRGB(x, y, new Color(R, G, B).getRGB());
        }
    }
    this.image Procesada = rgb;
    return rgb;
}
```

```java
//CMY --> CMYK
public BufferedImage convertirCMYaCMYK() {
    int w = imagenProcesada.getWidth();
    int h = imagen Procesada.getHeight();
    BufferedImage cmyk = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagenProcesada.getRGB(x, y));
            double C = c.get Red() / 255.0;
            double M = c.getGreen() / 255.0;
            double Y = c.getBlue() / 255.0;
            //calcular K
            double K = Math.min(C, Math.min(M, Y));
            double C_ = (K < 1.0) ? (C - K) / (1 - K) : 0;
            double M_ = (K < 1.0) ? (M - K) / (1 - K) : 0;
            double Y_ = (K < 1.0) ? (Y - K) / (1 - K) : 0;
            int R = (int)(C_ * 255);
            int G = (int)(M_ * 255);
            int B = (int)(Y_ * 255);

            int kGray = (int)(K * 255);
            Color cmykColor = new Color(R, G, kGray);

            cmyk.setRGB(x, y, cmykColor.getRGB());
        }
    }
    this.image Procesada = cmyk;
    return cmyk;
}
```

```java
//RGB --> YIQ
public BufferedImage convertirRGBaYIQ() {
    int w = imagen Original.getWidth();
    int h = imagen Original.getHeight();
    BufferedImage yiq = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagen Original.getRGB(x, y));
            double r = c.getRed() / 255.0;
            double g = c.getGreen() / 255.0;
            double b = c.getBlue() / 255.0;

            double Y = 0.299 * r + 0.587 * g + 0.114 * b;
            double I = 0.596 * r - 0.274 * g - 0.322 * b;
            double Q = 0.211 * r - 0.523 * g + 0.312 * b;

            int R = (int)(Math.min(1, Math.max(0, Y)) * 255);
            int G = (int)(Math.min(1, Math.max(0, I + 0.5)) * 255);
            int B = (int)(Math.min(1, Math.max(0, Q + 0.5)) * 255);

            yiq.setRGB(x, y, new Color(R, G, B).getRGB());
        }
    }
    this.imagenProcesada = yiq;
    return yiq;
}
```

```java
//YIQ --> RGB
public BufferedImage convertirYIQaRGB() {
    int w = imagenProcesada.getWidth();
    int h = imagenProcesada.getHeight();
    BufferedImage rgb = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagenProcesada.getRGB(x, y));
            double Y = c.getRed() / 255.0;
            double I = (c.getGreen() / 255.0) - 0.5;
            double Q = (c.getBlue() / 255.0) - 0.5;

            double r = Y + 0.956 * I + 0.621 * Q;
            double g = Y - 0.272 * I - 0.647 * Q;
            double b = Y - 1.106 * I + 1.703 * Q;

            int R = (int)Math.min(255, Math.max(0, r * 255));
            int G = (int)Math.min(255, Math.max(0, g * 255));
            int B = (int)Math.min(255, Math.max(0, b * 255));

            rgb.setRGB(x, y, new Color(R, G, B).getRGB());
        }
    }
    this.imagenProcesada = rgb;
    return rgb;
}
```

```java
//RGB --> HSI
public BufferedImage convertirRGBaHSI() {
    int w = imagenOriginal.getWidth();
    int h = imagenOriginal.getHeight();
    BufferedImage hsi = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagenOriginal.getRGB(x, y));
            double r = c.getRed() / 255.0;
            double g = c.getGreen() / 255.0;
            double b = c.getBlue() / 255.0;

            double num = 0.5 * ((r - g) + (r - b));
            double den = Math.sqrt((r - g) * (r - g) + (r - b) * (g - b));
            double theta = Math.acos(num / (den + 1e-9));

            double H = (b <= g) ? theta : (2 * Math.PI - theta);
            double S = 1 - (3 / (r + g + b + 1e-9)) * Math.min(r, Math.min(g, b));
            double I = (r + g + b) / 3.0;

            int R = (int)(H / (2 * Math.PI) * 255);
            int G = (int)(S * 255);
            int B = (int)(I * 255);

            hsi.setRGB(x, y, new Color(R, G, B).getRGB());
        }
    }
    this.imagenProcesada = hsi;
    return hsi;
}
```

```java
//HSI --> RGB
public BufferedImage convertirHSIaRGB() {
    int w = imagenProcesada.getWidth();
    int h = imagenProcesada.getHeight();
    BufferedImage rgb = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagenProcesada.getRGB(x, y));
            double H = (c.getRed() / 255.0) * 2 * Math.PI;
            double S = c.getGreen() / 255.0;
            double I = c.getBlue() / 255.0;
            double r=0,g=0,b=0;

            if (H < 2 * Math.PI / 3) {
                b = I * (1 - S);
                r = I * (1 + (S * Math.cos(H)) / Math.cos(Math.PI / 3 - H));
                g = 3 * I - (r + b);
            } else if (H < 4 * Math.PI / 3) {
                H = H - 2 * Math.PI / 3;
                r = I * (1 - S);
                g = I * (1 + (S * Math.cos(H)) / Math.cos(Math.PI / 3 - H));
                b = 3 * I - (r + g);
            } else {
                H = H - 4 * Math.PI / 3;
                g = I * (1 - S);
                b = I * (1 + (S * Math.cos(H)) / Math.cos(Math.PI / 3 - H));
                r = 3 * I - (g + b);
            }

            int R = (int)Math.min(255, Math.max(0, r * 255));
            int G = (int)Math.min(255, Math.max(0, g * 255));
            int B = (int)Math.min(255, Math.max(0, b * 255));

            rgb.setRGB(x, y, new Color(R, G, B).getRGB());
        }
    }
    this.imagenProcesada = rgb;
    return rgb;
}
```

```java
//RGB --> HSV
public BufferedImage convertirRGBaHSV() {
    int w = imagenOriginal.getWidth();
    int h = imagenOriginal.getHeight();
    BufferedImage hsvImg = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            Color c = new Color(imagenOriginal.getRGB(x, y));
            float[] hsv = Color.RGBtoHSB(c.getRed(), c.getGreen(), c.getBlue(),
            null);

            int H = (int)(hsv[0] * 255);
            int S = (int)(hsv[1] * 255);
            int V = (int)(hsv[2] * 255);

            hsvImg.setRGB(x, y, new Color(H, S, V).getRGB());
        }
    }
    this.imagenProcesada = hsvImg;
    return hsvImg;
}

//HSV --> RGB
public BufferedImage convertirHSVaRGB() {
    int w = imagenProcesada.getWidth();
    int h = imagenProcesada.getHeight();
    BufferedImage rgbImg = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            Color c = new Color(imagenProcesada.getRGB(x, y));
            float H = c.getRed() / 255f;
            float S = c.getGreen() / 255f;
            float V = c.getBlue() / 255f;

            int rgb = Color.HSBtoRGB(H, S, V);
            rgbImg.setRGB(x, y, rgb);
        }
    }
    this.imagenProcesada = rgbImg;
    return rgbImg;
}
```

```java
//RGB --> lab
public BufferedImage convertirRGBaLab() {
    int w = imagenOriginal.getWidth();
    int h = imagenOriginal.getHeight();
    BufferedImage labImg = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    //RGB ---> LMS
    double[][] rgb2lms = {
        {0.3811, 0.5783, 0.0402},
        {0.1967, 0.7244, 0.0782},
        {0.0241, 0.1288, 0.8444}
    };

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagenOriginal.getRGB(x, y));
            double r = c.getRed() / 255.0;
            double g = c.getGreen() / 255.0;
            double b = c.getBlue() / 255.0;

            //RGB ---> LMS
            double L = rgb2lms[0][0]*r + rgb2lms[0][1]*g + rgb2lms[0][2]*b;
            double M = rgb2lms[1][0]*r + rgb2lms[1][1]*g + rgb2lms[1][2]*b;
            double S = rgb2lms[2][0]*r + rgb2lms[2][1]*g + rgb2lms[2][2]*b;

            // Evitar log(0)
            L = Math.max(L, 1e-9);
            M = Math.max(M, 1e-9);
            S = Math.max(S, 1e-9);

           double lLog = Math.log10(L);
           double mLog = Math.log10(M);
           double sLog = Math.log10(S);

            //LMS ---> lab
            double Lp = (1.0/Math.sqrt(3)) * (lLog + mLog + sLog);
            double alpha = (1.0/Math.sqrt(6)) * (lLog + mLog - 2*sLog);
            double beta  = (1.0/Math.sqrt(2)) * (lLog - mLog);

            //limitar
            int R = (int)(Math.min(1, Math.max(0, (Lp + 1) / 2.0)) * 255);
            int G = (int)(Math.min(1, Math.max(0, (alpha + 1) / 2.0)) * 255);
            int B = (int)(Math.min(1, Math.max(0, (beta + 1) / 2.0)) * 255);

            labImg.setRGB(x, y, new Color(R, G, B).getRGB());
       }
    }
    this.imagenProcesada = labImg;
    return labImg;
}
```

```java
//lab --> RGB
public BufferedImage convertirLabaRGB() {
    int w = imagenProcesada.getWidth();
    int h = imagenProcesada.getHeight();
    BufferedImage rgbImg = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);

    //LMS ---> RGB
    double[][] lms2rgb = {
        { 4.4679, -3.5873,  0.1193},
        {-1.2186,  2.3809, -0.1624},
        { 0.0497, -0.2439,  1.2045}
    };

    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            Color c = new Color(imagenProcesada.getRGB(x, y));

            //lab
            double Lp = (c.getRed() / 255.0) * 2 - 1;
            double alpha = (c.getGreen() / 255.0) * 2 - 1;
            double beta  = (c.getBlue() / 255.0) * 2 - 1;

            //lab ---> LMS
            double lLog = (1/Math.sqrt(3.0))*Lp + (1/Math.sqrt(6.0))*alpha +
            (1/Math.sqrt(2.0))*beta;
            double mLog = (1/Math.sqrt(3.0))*Lp + (1/Math.sqrt(6.0))*alpha -
            (1/Math.sqrt(2.0))*beta;
            double sLog = (1/Math.sqrt(3.0))*Lp - (2/Math.sqrt(6.0))*alpha;

            //regresar a LMS
            double L = Math.pow(10, lLog);
            double M = Math.pow(10, mLog);
            double S = Math.pow(10, sLog);

            //LMS ---> RGB
            double r = lms2rgb[0][0]*L + lms2rgb[0][1]*M + lms2rgb[0][2]*S;
            double g = lms2rgb[1][0]*L + lms2rgb[1][1]*M + lms2rgb[1][2]*S;
            double b = lms2rgb[2][0]*L + lms2rgb[2][1]*M + lms2rgb[2][2]*S;

            int R = (int)Math.min(255, Math.max(0, r * 255));
            int G = (int)Math.min(255, Math.max(0, g * 255));
            int B = (int)Math.min(255, Math.max(0, b * 255));

            rgbImg.setRGB(x, y, new Color(R, G, B).getRGB());
        }
    }
    this.imagenProcesada = rgbImg;
    return rgbImg;
}
```

```java
//transferencia de color entre imagenes
public BufferedImage transferirColor(BufferedImage referencia) {
    int w = imagenOriginal.getWidth();
    int h = imagenOriginal.getHeight();

    //a lab
    double[][][] labOriginal = convertirRGBaLabArray(imagenOriginal);
    double[][][] labReferencia = convertirRGBaLabArray(referencia);

    //medias y desviaciones por canal
    double[] meanO = calcularMedia(labOriginal);
    double[] stdO  = calcularDesv(labOriginal, meanO);

    double[] meanR = calcularMedia(labReferencia);
    double[] stdR  = calcularDesv(labReferencia, meanR);

    BufferedImage result = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
     //crear imagen resultado

    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            double L = labOriginal[y][x][0];
            double a = labOriginal[y][x][1];
            double b = labOriginal[y][x][2];

            L = (L - meanO[0]) / stdO[0];
            a = (a - meanO[1]) / stdO[1];
            b = (b - meanO[2]) / stdO[2];

           L = L * stdR[0] + meanR[0];
           a = a * stdR[1] + meanR[1];
           b = b * stdR[2] + meanR[2];
           //lab --> RGB
           Color rgb = LabToRGB(L, a, b);
           result.setRGB(x, y, rgb.getRGB());
        }
    }

    this.imagenProcesada = result;
    return result;
}
```

Con este método para realizar la transferencia de color, se requiere para su funcionamiento calcular la media y la varianza, cosa que se realiza en la sección de “utilizades” dentro de la lógica, se implementa de la siguiente manera:

```
 private double[] calcularMedia(double[][][] lab) {
     double L=0,a=0,b=0;
     int h = lab.length, w = lab[0].length;
     int n = w*h;

     for (int y=0;y<h;y++){
         for(int x=0;x<w;x++){
             L += lab[y][x][0];
             a += lab[y][x][1];
             b += lab[y][x][2];
         }
     }
     return new double[]{L/n, a/n, b/n};
 }
private double[] calcularDesv(double[][][] lab, double[] mean) {
    double L=0,a=0,b=0;
    int h = lab.length, w = lab[0].length;
    int n = w*h;

    for (int y=0;y<h;y++){
        for(int x=0;x<w;x++){
            L += Math.pow(lab[y][x][0]-mean[0],2);
            a += Math.pow(lab[y][x][1]-mean[1],2);
            b += Math.pow(lab[y][x][2]-mean[2],2);
        }
    }
    return new double[]{
        Math.sqrt(L/n),
        Math.sqrt(a/n),
        Math.sqrt(b/n)
    };
}
```

Conclusión

A manera de conclusión digo que realizar las conversiones entre modelos de color es útil en diferentes aspectos dependiendo el contexto, tiene su nivel de complejidad, se me dificultó implementar el modelo Cie LAB especialmente porque es un modelo que requiere conversiones intermedias para realizarse pero el utilizar este modelo para la transferencia de color fue demasiado interesante y ver la imagen "recoloreada" con los colores obtenidos desde otra imagen fue un proceso que conllevó análisis y el resultado es demasiado satisfactorio.

Referencias

GeeksforGeeks. (2025, 23 julio). *What is HSI Color Space?* GeeksforGeeks. https://geeksforgeeks.org/electronics-engineering/what-is-hsi-color-space/

Modelos de colores (RGB a YIQ y viceversa) - Transformaciones Lineales. (s. f.). Rincón Matemático. https://foro.rinconmatematico.com/index.php?topic=117097.0

Reinhart, E., & Gooch, B. (2001). *Color Transfer between Images*. https://researchgate.net/publication/220518215_Color_Transfer_between_Images

Trost, S. (s. f.). *Modelos de color: CMY y CMYK*. https://es.sttmedia.com/modelo-de-color-cmyk

Wikipedia contributors. (2025, 30 agosto). *Image color transfer*. Wikipedia. https://en.wikipedia.org/wiki/Image_color_transfer