Taller de Procesamiento de Imágenes - Colores

Parte I: Conversión Manual de Espacios de Color

Fórmulas de conversión RGB → HSV:

```
1. Normalizar RGB a [0,1]:
   r = R/255, g = G/255, b = B/255

2. Calcular:
   Cmax = max(r, g, b)
   Cmin = min(r, g, b)
   Δ = Cmax - Cmin

3. Hue (H):
   Si Δ = 0: H = 0
   Si Cmax = r: H = 60° × (((g-b)/Δ) mod 6)
   Si Cmax = g: H = 60° × ((b-r)/Δ + 2)
   Si Cmax = b: H = 60° × ((r-g)/Δ + 4)
   H_opencv = H / 2  (OpenCV usa [0,180] en vez de [0,360])

4. Saturation (S):
   Si Cmax = 0: S = 0
   Si no: S = Δ/Cmax
   S_opencv = S * 255

5. Value (V):
   V = Cmax
   V_opencv = V * 255
```

Fórmulas de conversión HSV → RGB:

```
1. Desnormalizar:
   H = H_opencv * 2
   S = S_opencv / 255.0
   V = V_opencv / 255.0

2. Calcular:
   C = V × S
   X = C × (1 - |((H/60°) mod 2) - 1|)
   m = V - C

3. Según el rango de H:
   0° ≤ H < 60°:   (r',g',b') = (C, X, 0)
   60° ≤ H < 120°:  (r',g',b') = (X, C, 0)
   120° ≤ H < 180°: (r',g',b') = (0, C, X)
   180° ≤ H < 240°: (r',g',b') = (0, X, C)
   240° ≤ H < 300°: (r',g',b') = (X, 0, C)
   300° ≤ H < 360°: (r',g',b') = (C, 0, X)

4. Convertir a [0,255]:
   R = (r' + m) × 255
   G = (g' + m) × 255
   B = (b' + m) × 255
```

EJERCICIO 1: Implementar Conversión RGB → HSV

Objetivo: Convertir una imagen RGB a HSV píxel por píxel implementando las fórmulas.

Código base proporcionado:

```
void ejercicio1_rgb_a_hsv() {
    Mat img_bgr = imread("imagen.jpg");
    if (img_bgr.empty()) {
        cout << "Error: No se pudo cargar la imagen" << endl;
        return;
    }

    int rows = img_bgr.rows;
    int cols = img_bgr.cols;

    Mat img_hsv(rows, cols, CV_8UC3);

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            Vec3b pixel_bgr = img_bgr.at<Vec3b>(i, j);

            // Obtener valores BGR y normalizar

            // Calcular Cmax, Cmin, Delta

            // Calcular Hue (H)

            // Calcular Saturation (S)

            // Calcular Value (V)

            // Asignar valores HSV al píxel=

        }
    }

    imshow("Original BGR", img_bgr);
    imshow("HSV (Manual)", img_hsv);

    waitKey(0);
    destroyAllWindows();
}
```

EJERCICIO 2: Modificar Saturación Manualmente

Objetivo: Aumentar la saturación de una imagen trabajando directamente con los valores numéricos.

Código base proporcionado:

```
void ejercicio2_modificar_saturacion() {
    Mat img_bgr = imread("imagen.jpg");
    int rows = img_bgr.rows;
    int cols = img_bgr.cols;

    Mat img_hsv(rows, cols, CV_8UC3);

    // TODO: Copiar código de conversión BGR→HSV aquí
    // ... (del ejercicio 1)


    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            Vec3b pixel_hsv = img_hsv.at<Vec3b>(i, j);

            // PASO 1: Obtener valores H, S, V

            // PASO 2: Multiplicar S por 1.5 (sin exceder 255)

            // PASO 3: Asignar nuevos valores

        }
    }

    Mat img_resultado(rows, cols, CV_8UC3);

    // TODO: Implementar conversión HSV → BGR
    // Usa las fórmulas inversas explicadas en la teoría


    imshow("Original", img_bgr);
    imshow("Saturación Aumentada", img_resultado);

    waitKey(0);
    destroyAllWindows();
}
```

Parte II: K-Means Desde Cero

Algoritmo K-Means:

```
Entrada: Datos (píxeles), K (número de clusters)

1. Inicialización:
   - Seleccionar K centroides aleatorios

2. Asignación:
   Para cada punto de datos:
     - Calcular distancia a cada centroide
     - Asignar al centroide más cercano

3. Actualización:
   Para cada cluster:
     - Recalcular centroide como promedio de puntos asignados

4. Repetir pasos 2-3 hasta convergencia o max_iteraciones
```

Distancia Euclidiana en RGB:

```
d = sqrt((R1-R2)² + (G1-G2)² + (B1-B2)²)
```

EJERCICIO 3: Implementar K-Means Completo

Objetivo: Implementar K-Means desde cero sin usar `cv::kmeans()`.

Código base proporcionado:

```
struct Pixel {
    double r, g, b;
    Pixel() : r(0), g(0), b(0) {}
    Pixel(double r_, double g_, double b_) : r(r_), g(g_), b(b_) {}
};

double distancia_euclidiana(const Pixel& p1, const Pixel& p2) {
    // TODO: Implementar;
    return 0;
}

void ejercicio3_kmeans_manual(int K = 5) {
    Mat img_bgr = imread("imagen.jpg");
    if (img_bgr.empty()) return;

    // Redimensionar para acelerar
    Mat img_small;
    resize(img_bgr, img_small, Size(160, 120));

    int rows = img_small.rows;
    int cols = img_small.cols;
    int total_pixels = rows * cols;

    cout << "Procesando " << total_pixels << " píxeles con K=" << K << endl;

    // TODO: PASO 1 - Crear array de píxeles
    // Almacenar todos los píxeles en un vector

    // TODO: PASO 2 - Inicializar K centroides aleatorios

    // TODO: PASO 3 - Array para almacenar asignaciones
    // Cada píxel se asigna a un cluster [0, K-1]


    // TODO: PASO 4 - Iterar K-Means
    int max_iteraciones = 20;

    for (int iter = 0; iter < max_iteraciones; iter++) {
        cout << "Iteración " << (iter + 1) << "/" << max_iteraciones << endl;

        // PASO 4a: Asignar cada píxel al centroide más cercano

        // PASO 4b: Recalcular centroides
        // Crear arrays para sumar RGB de cada cluster

    }

    // TODO: PASO 5 - Crear imagen cuantizada
    // Reemplazar cada píxel por el color de su centroide


    // TODO: PASO 6 - Crear paleta de colores


    imshow("Original", img_small);
    // imshow("K-Means Manual K=" + to_string(K), img_quantized);
    // imshow("Paleta", paleta);

    waitKey(0);
    destroyAllWindows();
}
```

Parte III: Constancia de Color Manual

Gray World - Fórmulas:

```
1. Calcular promedio de cada canal:
   avg_R = (Σ R_i) / N
   avg_G = (Σ G_i) / N
   avg_B = (Σ B_i) / N

2. Calcular promedio gris:
   gray_avg = (avg_R + avg_G + avg_B) / 3

3. Calcular factores de escala:
   scale_R = gray_avg / avg_R
   scale_G = gray_avg / avg_G
   scale_B = gray_avg / avg_B

4. Aplicar a cada píxel:
   R_nuevo = min(R_viejo × scale_R, 255)
   G_nuevo = min(G_viejo × scale_G, 255)
   B_nuevo = min(B_viejo × scale_B, 255)
```

EJERCICIO 4: Implementar Gray World

Código base proporcionado:

```
void ejercicio4_gray_world() {
    Mat img_bgr = imread("imagen.jpg");
    if (img_bgr.empty()) return;

    int rows = img_bgr.rows;
    int cols = img_bgr.cols;
    int total_pixels = rows * cols;

    // TODO: PASO 1 - Calcular suma de cada canal

    // TODO: PASO 2 - Calcular promedios

    // TODO: PASO 3 - Calcular promedio gris

    // TODO: PASO 4 - Calcular factores de escala

    // TODO: PASO 5 - Crear imagen corregida

    imshow("Original", img_bgr);
    // imshow("Gray World", img_resultado);

    waitKey(0);
    destroyAllWindows();
}
```

Parte IV: Calibración Radiométrica Manual

Corrección Gamma:

```
Formula: pixel_out = 255 × (pixel_in / 255)^gamma

- gamma < 1: Aclara la imagen
- gamma = 1: Sin cambio
- gamma > 1: Oscurece la imagen

Ejemplo con gamma = 0.5:
  pixel_in = 100
  pixel_out = 255 × (100/255)^0.5 = 160
```

Corrección de Viñeteo:

```
1. Calcular centro de imagen:
   cx = ancho / 2
   cy = alto / 2

2. Para cada píxel (x, y):
   - Distancia al centro: d = sqrt((x-cx)² + (y-cy)²)
   - Distancia normalizada: d_norm = d / d_max
   - Factor de corrección: f = 1 / (1 - k × d_norm²)
   - Píxel corregido = min(píxel_original × f, 255)

donde k es el coeficiente de viñeteo (típicamente 0.3-0.5)
```

EJERCICIO 6: Implementar Corrección Gamma

Código base proporcionado:

```
void ejercicio6_gamma(double gamma = 1.5) {
    Mat img_bgr = imread("imagen.jpg");
    if (img_bgr.empty()) return;

    int rows = img_bgr.rows;
    int cols = img_bgr.cols;

    // TODO: PASO 1 - Crear tabla de lookup (para eficiencia)
    // Pre-calcular la transformación para todos los valores 0-255

    // TODO: PASO 2 - Aplicar transformación a cada píxel

    imshow("Original", img_bgr);
    // imshow("Gamma = " + to_string(gamma), img_resultado);

    waitKey(0);
    destroyAllWindows();
}
```

EJERCICIO 7: Implementar Corrección de Viñeteo

Código base proporcionado:

```
void ejercicio7_vignette(double k = 0.4) {
    Mat img_bgr = imread("imagen.jpg");
    if (img_bgr.empty()) return;

    int rows = img_bgr.rows;
    int cols = img_bgr.cols;

    // TODO: PASO 1 - Calcular centro de la imagen

    // TODO: PASO 2 - Calcular distancia máxima (a la esquina)

    // TODO: PASO 3 - Aplicar corrección píxel por píxel

    imshow("Original", img_bgr);
    // imshow("Viñeteo Corregido k=" + to_string(k), img_resultado);

    waitKey(0);
    destroyAllWindows();
}
```

Funciones de OpenCV PERMITIDAS

```
// PERMITIDAS (I/O básico):
Mat img = imread("imagen.jpg");
imshow("titulo", img);
waitKey(0);
imwrite("salida.jpg", img);
resize(img, img_small, Size(w, h));

// PERMITIDAS (Creación de matrices):
Mat img_nueva(rows, cols, CV_8UC3);
img.at<Vec3b>(i, j) = Vec3b(b, g, r);

// PERMITIDAS (Matemáticas):
sqrt(), pow(), sin(), cos(), max(), min()

// PROHIBIDAS:
cvtColor() - Implementa las fórmulas tú mismo
inRange() - Compara valores manualmente
kmeans() - Implementa el algoritmo completo
LUT() - Aplica transformaciones en bucle
split() / merge() - Accede a canales manualmente
```