

Imagenes digitales

Tomas Lopez Perez

April 29, 2024

1 Introducción

El procesamiento de imágenes digitales se utiliza para mejorar la calidad de las imágenes y extraer información valiosa de ellas. Esto puede implicar técnicas como la mejora de la imagen, la segmentación de la imagen y la transformación de la imagen. Estas técnicas se utilizan en una variedad de campos [HST71], incluyendo el reconocimiento de patrones y la codificación eficiente de imágenes. Un aspecto importante del procesamiento de imágenes digitales es el manejo de imágenes RAW. Las imágenes RAW son datos sin procesar directamente del sensor de la cámara. Estas imágenes ofrecen una mayor flexibilidad en el procesamiento posterior porque contienen más detalles y menos artefactos de compresión que las imágenes procesadas. El proceso para extraer una imagen RAW de un filtro Bayer es un paso crucial en el flujo de trabajo de procesamiento de imágenes. Este proceso incluye el ajuste del balance de blancos y la conversión al espacio de color RGB.

2 Fundamento teórico

2.1 Procesamiento de imágenes

El Procesamiento de imágenes es un campo de investigación muy extenso que involucra diversas áreas del conocimiento, ya que involucra diversos procesos, tales como: la adquisición, transmisión, representación y procesamiento. En términos generales el procesamiento de imágenes se utiliza para modificar una imagen con el fin de mejorar la apariencia visual para un observador y para resaltar convenientemente el contenido de la imagen de cara a la

percepción por parte de máquinas, es decir, se hace una manipulación de imágenes con objeto de producir nuevas imágenes que son mejores, en algún sentido. El procesamiento de imágenes comprende distintas técnicas que se utilizan para mejorar, restaurar e inclusive comprimir imágenes ya existentes; la implementación y desarrollo de dichas técnicas agregadas a sistemas ya existentes nos permiten editar imágenes de forma más sencilla y precisa [Alo18]. La Figura 1 muestra el flujo de las etapas del procesamiento digital de imágenes propuestas por Gonzalez en donde se establece como la etapa inicial la adquisición de la imagen digital, lo que se puede hacer mediante algún sistema de sensores que permitan digitalizar la imagen u obtener las imágenes de algún repositorio o colección. El conjunto de imágenes adquiridas conforma inicialmente la base de conocimiento, ya que para algunas aplicaciones basta con tener la información pura, sin embargo en la mayoría de las aplicaciones del procesamiento de imágenes, es necesario enriquecer la base del conocimiento con al menos las etapas de procesado y segmentación, lo cual requiere aplicar una serie de operadores que transformen las imágenes para resaltar características relevantes, en el caso del procesado, o separar los diferentes elementos que componen la imagen, en el caso de la segmentación, de modo que los resultados de estas dos etapas permiten tener una base del conocimiento más amplia y robusta. Respecto a la etapa de representación y descripción, las imágenes procesadas y/o segmentadas son usadas para extraer características más específicas sobre la imagen, tales como cantidad, dimensión, forma, posición, etc. de los objetos de interés en las imágenes, esta información también puede formar parte de la base del conocimiento, de

modo que en la ultima etapa, reconocimiento e interpretación, se puedan obtener respuestas a partir de las imágenes y la información generada por estas.

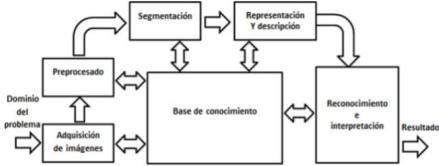


Figure 1: Etapas fundamentales del procesamiento digital de imágenes.

3 Metodología

Para llevar a cabo este trabajo se realiza la lectura de una imagen raw con el software MatLab el cual nos permite simplificar los pasos siguientes. Adquisición de la imagen: En este paso, se adquiere la imagen RAW directamente del sensor de la cámara. Esta imagen contiene todos los detalles capturados por el sensor y no ha sido alterada o comprimida.

Las imágenes RAW no tienen un valor de negro real, la cámara hace una estimación del valor de negro basándose en las intensidades de los pixeles enmascarados y se encuentra en los metadatos de la imagen, por lo tanto debemos de aplicar los valores que contiene la matriz (ColorInfo.BlackLevel) 2. multiplicando con la imagen original y después obtenemos el valor mayor , dividimos a la matriz original para obtener una imagen normalizada 3

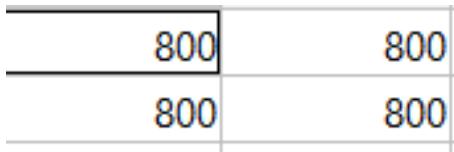


Figure 2: Negros reales

Ajuste del balance de blancos: El balance de blancos es un proceso que ajusta los colores de la imagen para que parezcan más naturales a nuestros ojos.

```
%Balance de negros
bayerNormalizado = double(subBayer) - BlackLevel;
max_valor = double(max(max(bayerNormalizado)));
bayerNormalizado = bayerNormalizado ./ max_valor;
```

Figure 3: Código aplicando el nivel de negros

Esto se hace ajustando los tonos de color en la imagen para que el color blanco se vea realmente blanco, y no tenga un tinte de color. La figura 5 es el código implementado, en este caso se realizó la multiplicación punto con la matriz 4 y después realizamos de nuevo la normalización.



Figure 4: Balance de blancos

```
%Balance de blancos
bayerBalanceado = im2uint16(bayerNormalizado .* WhiteBalance);
max_valor = double(max(max(bayerBalanceado)));
bayerBalanceado = double(bayerBalanceado) ./ max_valor;
```

Figure 5: Código implementado para el ajuste de blancos

Patrón bayer: Una vez ajustado el balance de blancos, se separan los mosaicos del patrón bayer RGGB. El código 6 realiza la implementación de la separación de los canales , Rojo, Verde, Azul, Los resultados los podemos ver en las figuras 16 17 18

Interpolación bayer : Generar la interpolación en cada mosaico para obtener valores perdidos de cada pixel en este caso se aplicó la interpolación lineal en bordes y la bilineal en el resto de los mosaicos. Para los canales se implementó dos Interpolaciones, la interpolación lineal con la cual tenemos que obtener

```

function bayerRojo = separar_rojo(bayer)
    %selecciona los elementos en las filas impares y columnas impares
    rojo= bayer(1:2:end, 1:2:end);
    [filas, columnas] = size(rojo);
    bayerRojo = zeros(filas * 2, columnas * 2, 'like', rojo);
    bayerRojo(1:2:end, 1:2:end) = rojo;
end

function bayerVerde = separarVerde(bayer)
    %filas impares y columnas pares
    verde1 = bayer(1:2:end, 2:2:end);
    [filas, columnas] = size(verde1);
    bayerVerde = zeros(filas * 2, columnas * 2, 'like', verde1);
    bayerVerde(1:2:end, 2:2:end) = verde1;

    %Filas pares y Columnas impares
    verde2 = bayer(2:2:end, 1:2:end);
    bayerVerde(2:2:end, 1:2:end) = verde2;
    %bayerVerde(2:2:end, 1:2:end) = bayerVerde2C;
end

function bayerAzul = separarAzul(bayer)
    %Seleccionando Filas pares y Columnnas impares pero aumentando 2
    azul = bayer(2:2:end, 2:2:end);
    [filas, columnas] = size(azul);
    bayerAzul= zeros(filas * 2, columnas * 2, 'like', azul);
    bayerAzul(2:2:end, 2:2:end) = azul;
end

```

```

function bayer = interpolacion_bilineal(bayer, posiciones)
    % Extraer las coordenadas de las posiciones
    rows = posiciones(:, 1);
    cols = posiciones(:, 2);

    % Calcular los índices de los elementos en las posiciones superiores e inferiores
    top_left_indices = sub2ind(size(bayer), rows-1, cols-1);
    top_right_indices = sub2ind(size(bayer), rows-1, cols+1);
    bottom_left_indices = sub2ind(size(bayer), rows+1, cols-1);
    bottom_right_indices = sub2ind(size(bayer), rows+1, cols+1);

    % Obtener los valores en las posiciones superiores e inferiores
    y1_top_left = bayer(top_left_indices);
    y2_top_right = bayer(top_right_indices);
    y1_bottom_left = bayer(bottom_left_indices);
    y2_bottom_right = bayer(bottom_right_indices);

    % Interpolación lineal en las posiciones superiores e inferiores
    x1 = cols-1;
    x2 = cols+1;
    x = cols;

    y_first = y1_top_left + (x - x1) .* (y2_top_right - y1_top_left) ./ (x2 - x1);
    y_second = y1_bottom_left + (x - x1) .* (y2_bottom_right - y1_bottom_left) ./ (x2 - x1);

    x = cols;
    % Interpolación bilineal
    y_second = y_first + (x - x1) .* (y_second - y_first) ./ (x2 - x1);
    % Asignar los valores interpolados a las posiciones actuales en la matriz
    bayer(sub2ind(size(bayer), rows, cols)) = y_second;
end

```

Figure 6: Código para la separación RGGB

los 2 vecinos más cercanos y la bilineal que debemos tomar los 4 vecinos más cercanos.

```

function valores = interpolar(valores)
    % Encontrar los índices de los valores que no son cero
    indices_no_ceros = find(valores ~= 0);
    valores_no_ceros = valores(indices_no_ceros);
    % Encontrar los índices de los valores que son cero
    indices_ceros = find(valores == 0);
    % Calcular la interpolación lineal usando interp1 con los mismos puntos de interpolación
    valores_interp = interp1(indices_no_ceros, valores_no_ceros, indices_ceros, 'linear');
    % Asignar los valores interpolados al vector original
    valores(indices_ceros) = valores_interp;
end

```

Figure 7: Interpolación Lineal

Conversión al espacio de color RGB: Aquí se hace uso de la matriz de transformación CameraToRGB que lo obtenemos de la imagen raw, para convertir del espacio lineal de la cámara al espacio de color lineal RGB. 10

Corrección Gamma: Compensar la falta de luminosidad para obtener una imagen con mejor balance de colores. 11

4 Resultados

Para las pruebas se obtuvo una imagen RAW Figure 12 de la cual se extrajeron varias características

como por ejemplo el filtro con el que fue capturada la imagen en este caso es el filtro RGGB figure 13

Se aplica el normalizado de la imagen, restamos los valores de los pixeles con el esquema RGGB el cual contiene valores (800, 800, 800, 800) para posteriormente dividirlo por el máximo valor dentro de la matriz. figure 14 , continuamos aplicando el ajuste de balance de blancos el cual reliza la multiplicación con los valores del esquema bayer RGGB escalando los valores del filtro verde a 1 teniendo (2.264, 1; 1, 1.832) que nos da como resultado figura 15

Separamos los 3 canales con forma al filtro bayer RGGB. El canal rojo figura 16 , canal verde 17 y el canal Azul 18.

Continuamos uniendo los canales en una sola matriz para después multiplicarla por la matriz de color RGB figura 21 para que nos de como resultado la a imagen a color 20 21

5 Conclusiones

La aplicación de color a una imagen raw implica ajustar el balance de blancos, separar los canales de color y realizar interpolación para completar los píxeles perdidos. Estos procesos, realizados con herramientas como matlab, aseguran una reproducción precisa

```

function bayer = interpolacion_bilinealcruz(bayer, posiciones)
    % Extraer las coordenadas de las posiciones
    rows = posiciones(:, 1);
    cols = posiciones(:, 2);
    % Calcular los indices de los elementos en las posiciones superiores e inferiores
    top_left_indices = sub2ind(size(bayer), rows, cols-1);
    top_right_indices = sub2ind(size(bayer), rows-1, cols);
    bottom_left_indices = sub2ind(size(bayer), rows+1, cols);
    bottom_right_indices = sub2ind(size(bayer), rows, cols+1);

    % Obtener los valores en las posiciones superiores e inferiores
    y1_top_left = bayer(top_left_indices);
    y2_top_right = bayer(top_right_indices);
    y1_bottom_left = bayer(bottom_left_indices);
    y2_bottom_right = bayer(bottom_right_indices);

    % Interpolación lineal en las posiciones superiores e inferiores
    x1 = cols-1;
    x2 = cols;
    x = cols;
    y1_first = y1_top_left + (x - x1) .* (y2_top_right - y1_top_left) ./ (x2 - x1);
    x1 = cols;
    x2 = cols+1;
    y1_second = y1_bottom_left + (x - x1) .* (y2_bottom_right - y1_bottom_left) ./ (x2 - x1);
    % Interpolación bilineal
    y1_second = y1_first + (x - x1) .* (y1_second - y1_first) ./ (x2 - x1);
    % Asignar los valores interpolados a las posiciones actuales en la matriz
    bayer(sub2ind(size(bayer), rows, cols)) = y1_second;
end

```

Figure 9: Interpolacion Bilineal de cruz

```

[filas, columnas] = size(bayerRI);
bayer3D = cat(3, bayerRI, bayerVI, bayerAI);
bayer_reshaped = reshape(bayer3D, [], 3)';
rgb_reshaped = cameraTorgb * bayer_reshaped;
bayer3D = reshape(rgb_reshaped', filas, columnas, []);
bayerColor = generate_gamma(bayer3D);

```

Figure 10: Generar Imagen

del color y una representación visualmente atractiva. En resumen, permiten mejorar la calidad y apariencia de la imagen final, con ajustes precisos para obtener resultados óptimos.

References

- [HST71] T.S. Huang, W.F. Schreiber, and O.J. Tretiak. “Image processing”. In: *Proceedings of the IEEE* 59.11 (1971), pp. 1586–1609. DOI: 10.1109/PROC.1971.8491.
- [Alo18] Adan Antonio Alonso-Ramírez. *Implementación de la transformada de Hough en tecnología GPU*. 2018.

```

function imagenColor = generate_gamma(bayerColor)
    %gamma = 1/2.222;
    gamma = 0.7;
    a_max = max(bayerColor(:));
    bayerColor(:, :, 1) = bayerColor(:, :, 1) .^ gamma;
    % Corrección gamma para el canal verde
    bayerColor(:, :, 2) = bayerColor(:, :, 2) .^ gamma;
    % Corrección gamma para el canal azul
    bayerColor(:, :, 3) = bayerColor(:, :, 3). ^ gamma;
    %bayerColor = (double(bayerColor) / 255) .^ gamma * 255;
    %bayerColor = (bayerColor/a_max) .^ gamma;
    imagenColor = bayerColor;
end

```

Figure 11: Aplicar Gamma



Figure 12: Imagen raw

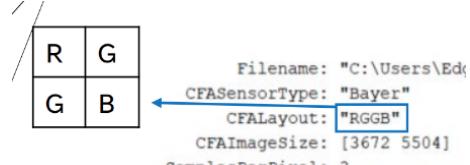


Figure 13: Filtro RGGB



Figure 14: Imagen raW Normalizada

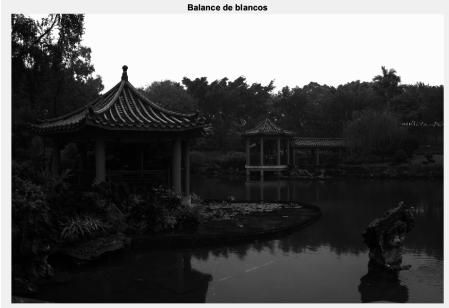


Figure 15: Imagen raw con ajustes de blancos

	1	2	3
1	1.6342	-0.4373	-0.1969
2	-0.1839	1.5024	-0.3185
3	0.0408	-0.4297	1.3889

Figure 19: Camera to RGB

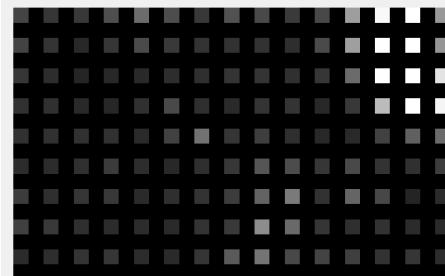


Figure 16: Canal Rojo



Figure 20: Imagen a color

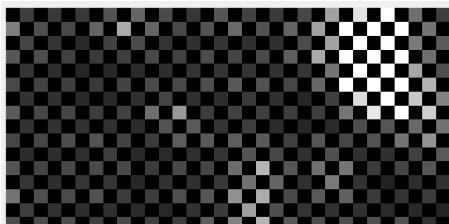


Figure 17: Canal Verde



Figure 21: Imagen color Gamma

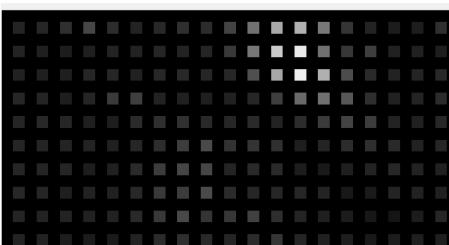


Figure 18: Canal Azul