

INTRODUCCIÓN AL PROCESAMIENTO ESPACIAL DE IMÁGENES

VERSIÓN EN DESARROLLO

JIMY ALEXANDER CORTÉS OSORIO

FRANCISCO ALEJANDRO MEDINA AGUIRRE

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE CIENCIAS BÁSICAS
PEREIRA
2025

Dedicatoria

A nuestros estudiantes quienes han hecho de esta guía un documento de referencia.

Agradecimientos

A los programas de Ingeniería Mecatrónica e Ingeniería Física de la Universidad Tecnológica de Pereira, por sus aportes académicos a través de sus estudiantes: Ingeniero Alejandro Arroyabe, Ingeniero Daniel Felipe Socha, Ingeniero Juan Pablo Peláez, Ingeniera María Camila Quintero y los profesores: Magíster Marlen Julieth Suárez, que realimentan permanentemente la construcción de este libro.

Al amigo eterno Magíster Andrew Murray Knott (QEPD) por sus aportes, disposición y ayuda desinteresada a la realización del presente documento.

Índice general

Dedicatoria	III
Agradecimientos	V
Introducción	XXI
1. Antecedentes	1
1.1. Antecedentes del Procesamiento de imágenes	1
1.1.1. Algunos momentos Históricos de la Fotografía	1
1.1.2. La primera fotografía a color de la Historia	2
1.1.3. Retoque de imágenes	4
1.1.4. Sobre la Transmisión de imágenes	6
1.2. Preguntas	8
1.2.1. Preguntas de comprensión:	8
1.2.2. Preguntas de aplicación y evaluación:	8
2. La Cámara	9
2.1. La luz	9
2.1.1. Naturaleza de la luz	9
2.1.2. El ojo Humano	10
2.2. La cámara	11
2.2.1. Formación del imagen	11
2.2.2. Formación de imagen en la cámara estenopeica	11
2.2.3. Formación de imágenes con lentes delgadas	13
2.3. Cámara Digital	16
2.4. Principio básico del color	17
2.4.1. Cámara con tres arreglos	18
2.4.2. Cámara con un solo arreglo	19
2.4.3. Ejemplo en Python: Separación de capas de color	21
2.5. Funciones	22
2.6. Preguntas	23
2.6.1. Preguntas sobre hechos del texto:	23
2.6.2. Preguntas de análisis y comprensión sobre el texto:	23
3. La Imagen	25
3.1. La imagen	25
3.2. La Imagen Digital	25
3.3. La Imagen a color	26
3.3.1. Ejemplo en Python: Creación de pixeles de colores	27
3.4. La Resolución Espacial	28
3.5. La profundidad del Color	29
3.6. Algunos Formatos Gráficos	30
3.6.1. JPEG	30

3.6.2. PNG	31
3.6.3. GIF	31
3.6.4. TIFF	32
3.6.5. BMP	32
3.6.6. RAW	33
3.7. Funciones	33
3.8. Preguntas	34
3.8.1. Preguntas de Comprensión	34
3.8.2. Preguntas de Aplicación y Evaluación	34
4. Ajuste y Ecualización	35
4.1. El Histograma de una imagen	35
4.1.1. Ejemplo en Python: Extracción de capas e Histogramas	38
4.1.2. El Brillo	40
4.1.3. El contraste	41
4.1.4.	42
4.1.5. Ejemplo en clase: Construcción del histograma	42
4.2. Normalización del Histograma	45
4.2.1. Ajuste por Potencia del Histograma	46
4.2.2. Ejemplo en clase: ajuste del histograma por Potencia	48
4.2.3. Ajuste Exponencial del Histograma	50
4.2.4. Ajuste Logarítmico del Histograma	50
4.2.5. Determinación de los Valores Em y EM	51
4.2.6. Ejemplo en Python: Ajuste de una imagen	52
4.3. Ecualización del histograma	56
4.3.1. Ejemplo en clase: Ecualización	56
4.3.2. Ejemplo en Clase: Ajuste por Ecualización	57
4.3.3. Ejemplo en Python: Ecualización de una imagen	59
4.4. Histograma por Especificación	63
4.4.1. Algoritmo Histograma por Especificación usando el Mayor Vecino más Cercano	63
4.4.2. Algoritmo Histograma por Especificación usando Interpolación Lineal	64
4.4.3. Ejemplo en Clase: Ajuste por Especificación	65
4.4.4. Ejemplo en Python: Ecualización por especificación	69
4.5. Funciones	72
4.6. Preguntas	72
4.6.1. Preguntas sobre hechos indicados en el texto:	72
4.6.2. Preguntas de análisis y comprensión con base en el texto:	72
4.6.3. Ejercicios numéricos	72
5. Modelos básicos del Color	75
5.1. El concepto del color	75
5.2. Modelos del Color	76
5.2.1. Modelo de color CIE XYZ	76
5.2.2. Modelo de color RGB	78
5.2.3. Modelo del color CMYK	79
5.2.4. Modelo de color HSV	80
5.2.5. Conversión de RGB a CMYK	81
5.2.6. Conversión de CMYK a RGB	82
5.2.7. Conversión de RGB a HSV	83
5.2.8. Conversión de HSV a RGB	85
5.2.9. Conversión de RGB a XYZ	86
5.2.10. Conversión de XYZ a RGB	88
5.2.11. Conversión de XYZ a Lab	90
5.2.12. Conversión de Lab a XYZ	93
5.2.13. Conversión de Lab a RGB	95
5.2.14. Conversión de RGB a Lab	95

5.3.	Ejemplo Segmentación	96
5.3.1.	Ejemplo en Python: Segmentación de la bandera usando el modelo RGB	98
5.3.2.	Ejemplo en Python: Segmentación de la bandera usando el modelo HSV	100
5.3.3.	Ejemplo en Python: Segmentación de la bandera usando el modelo CieLab	103
5.4.	Funciones	105
5.5.	Preguntas	106
5.5.1.	Preguntas directas sobre el texto	106
5.5.2.	Preguntas de análisis y compresión con base en el texto	106
5.5.3.	Preguntas de cálculos	107
6.	Conversión a escala de grises	109
6.1.	El problema	109
6.2.	Técnica del promedio	109
6.3.	Técnica del Midgray	110
6.4.	Técnica de la luminancia	111
6.5.	Conversión de escala de grises a color	112
6.5.1.	Ejemplo en Python: Segmentación por Color y Grises	112
6.6.	Funciones	115
6.7.	Preguntas	115
6.7.1.	Preguntas de Hechos Indicados en el Texto	115
6.7.2.	Preguntas de Comprensión con Base en el Texto	115
6.7.3.	Ejercicios Numéricos con Base en el Texto	117
7.	Binarización	119
7.1.	Técnicas globales	121
7.1.1.	Método global Iterativo	122
7.1.2.	Método global de la Entropía	123
7.1.3.	Método global de Otsu	124
7.1.4.	Comparación de los Métodos Globales Presentados	125
7.1.5.	Ejemplo en clase: Binarización Método Ridler	125
7.1.6.	Ejemplo en clase: Binarización Método de la Entropía	128
7.1.7.	Ejemplo en clase: Binarización Método Otsu	130
7.1.8.	Ejemplo Python	135
7.2.	Técnicas Adaptivas	137
7.2.1.	Método Adaptivo de Niblack	139
7.2.2.	Método Adaptivo de Sauvola	139
7.2.3.	Método Adaptivo de Bradley	140
7.2.4.	Comparación de los Métodos Adaptivos Presentados	142
7.2.5.	Ejemplo en clase: Binarización Método de Niblack	142
7.2.6.	Ejemplo en clase: Binarización Método de Sauvola	143
7.2.7.	Ejemplo en clase: Binarización Método de Bradley	144
7.2.8.	Ejemplo en Python: comparación de la binarización global y adaptativa	145
7.3.	Funciones	146
7.4.	Preguntas	147
7.4.1.	Preguntas sobre hechos indicados en el texto	147
7.4.2.	Preguntas de análisis y comprensión con base en el texto	147
7.4.3.	Ejercicios numéricos	148
8.	El Ruido	149
8.1.	El Ruido	149
8.2.	Fuentes de Ruido	149
8.2.1.	Ruido Impulsivo	149
8.2.2.	Ruido Gaussiano	151
8.2.3.	Ruido Speckle	152
8.3.	Evaluación Referencial del Filtrado de Imágenes	153
8.3.1.	Error Cuadrático Medio (MSE)	154
8.3.2.	Ejemplo Cálculo de MSE	155

8.3.3.	Relación Señal-Ruido de Pico (PSNR)	155
8.3.4.	Ejemplo Cálculo de PSNR	156
8.3.5.	Índice de Similitud Estructural (SSIM)	156
8.3.6.	Ejemplo Cálculo de SSIM	158
8.4.	Funciones	161
8.5.	Preguntas	161
8.5.1.	Preguntas sobre hechos del texto	161
8.5.2.	Preguntas de análisis y comprensión sobre el texto	161
8.5.3.	Problemas Numéricos sobre Ruido en Imágenes	162
9.	Filtros Espaciales	163
9.1.	Filtrado Espacial de Imágenes	163
9.2.	Operaciones Espaciales	163
9.2.1.	Correlación en Dos Dimensiones	163
9.2.2.	Convolución en Dos Dimensiones	164
9.2.3.	Ejemplo: evaluación de la Correlación y la Convolución	165
9.3.	Equivalencia entre Correlación y Convolución	166
9.3.1.	Ejemplo: evaluación de la Convolución usado la Correlación	167
9.4.	Solución	167
9.5.	Tipos de Relleno	168
9.5.1.	Relleno de Constante (Constant Padding)	168
9.5.2.	Relleno Cero (Zero Padding)	168
9.5.3.	Relleno Replicado (Replicate Padding)	169
9.5.4.	Relleno Simétrico (Symmetric Padding)	169
9.5.5.	Relleno Cílico (Circular Padding)	169
9.6.	Filtros de Suavizado (Pasa-Bajo)	170
9.6.1.	Filtro Aritmético	170
9.6.2.	Filtros de Orden	171
9.6.3.	Filtro Mediana	173
9.6.4.	Filtro Moda	175
9.6.5.	Filtro Desviación Estándar	175
9.6.6.	Filtro Entropía	177
9.6.7.	Ejemplo en Python: Filtros espaciales	177
9.6.8.	Filtro Gaussiano	180
9.6.9.	Ejemplo en Python: Filtros pasa bajo	182
9.7.	Filtros de Detector de Bordes (Pasa-Alto)	183
9.7.1.	El Gradiente	185
9.7.2.	El Laplaciano	187
9.7.3.	Filtro Laplaciano del Gaussiano	191
9.7.4.	Filtro Sharpen	194
9.7.5.	Filtro Sobel	196
9.7.6.	Filtro Prewitt	196
9.7.7.	Ejemplo en Python: Filtros pasa bajo	198
9.8.	Preguntas	201
9.8.1.	Preguntas sobre hechos indicados en el texto	201
9.8.2.	Preguntas de análisis y comprensión con base en el texto	201
9.8.3.	Ejercicios Numéricos	201
10.	Transformaciones Espaciales	203
10.1.	Método de Transformación Directo	204
10.2.	Método de Transformación Inverso	205
10.3.	Transformaciones en Sistema Cartesiano y Matricial	206
10.3.1.	Sistema de Coordenadas Cartesiano	206
10.3.2.	Sistema Matricial	207
10.3.3.	Interpretación de Coordenadas	207
10.3.4.	Aplicación de Transformaciones a Imágenes	207
10.4.	Transformaciones Geométricas Afines	207

10.4.1. Translación	208
10.4.2. Algoritmo Inverso de Translación	209
10.4.3. Escalamiento	212
10.4.4. Algoritmo Inverso de Escalamiento	214
10.4.5. Rotación	218
10.4.6. Algoritmo Inverso de Rotación	220
10.4.7. Ejemplo en Python: Transformaciones afines básicas	225
10.4.8. Cizallamiento	226
10.4.9. Algoritmo Inverso de cizallamiento	228
10.5. Transformación Proyectiva	231
10.5.1. Ejemplo en Python: Transformaciones proyectivas	233
10.6. Interpolaciones	235
10.6.1. Vecino más cercano	235
10.6.2. Interpolación bilineal	236
10.7. Funciones	239
10.8. Preguntas	240
10.8.1. Preguntas sobre hechos indicados en el texto	240
10.8.2. Preguntas de análisis y comprensión	240
10.8.3. Ejercicios numéricos	241
11. Operaciones Morfológicas Binarias	243
11.1. Conceptos Básicos de Teoría de Conjuntos y Lógica Binaria	243
11.1.1. Unión y el OR Lógico	243
11.1.2. Intersección y AND Lógico	244
11.1.3. Complemento y el NOT Lógico	245
11.2. El Elemento Estructurante	245
11.2.1. Elemento Estructurante Cuadrado	245
11.2.2. Elemento Estructurante Rectángulo	245
11.2.3. Elemento Estructurante Disco	245
11.2.4. Elemento Estructurante Cruz	247
11.2.5. Elemento Estructurante Línea	247
11.3. La Erosión	247
11.4. Algoritmo de Erosión	247
11.5. La Dilatación	248
11.5.1. Algoritmo de Dilatación	249
11.6. Dualidad entre Erosión y Dilatación	250
11.7. La Apertura Morfológica	252
11.7.1. Algoritmo Apertura	252
11.8. El Cerramiento Morfológico	254
11.8.1. Algoritmo de Cerramiento	254
11.8.2. Ejemplo en Python: segmentación mediante morfología	256
11.9. Funciones	258
11.10 Preguntas	259
12. Etiquetado	263
12.1. Análisis de componente conexos	263
12.2. Conectores bidimensionales	263
12.2.1. Conector 4	264
12.2.2. Conector 8	264
12.2.3. Proceso de Etiquetado	264
12.2.4. Ejemplo en clase: etiquetado de regiones	265
12.2.5. Ejemplo Python: Etiquetado de regiones	266
12.3. Funciones	268
12.4. Preguntas	268
12.4.1. Preguntas sobre hechos indicados en el texto	268
12.4.2. Preguntas de análisis y comprensión con base en el texto	269
12.4.3. Ejercicio numérico	269

13.Funciones de Python	271
Bibliografía	292

Índice de figuras

1.1. Inventores de la tecnología CCD. Willard S. Boyle (a la izquierda) y George E. Smith (a la derecha).	2
1.2. La revista TIME creó una lista de las 100 imágenes más influyentes jamás tomadas.	3
1.3.	4
1.4. Edición de imágenes por parte de la Unión Soviética. Imagen editada a la izquierda y original a la derecha.	5
1.5. Edición de imágenes por parte de Benito Mussolini. Imagen editada a la izquierda y original a la derecha.	5
1.6. Edición de imágenes por parte de Adolf Hitler. Imagen editada a la izquierda y original a la derecha.	6
1.7. (a) Bidwell, Shelford, The Board of Trustees of the Science Museum, London, 1881 (b) Primera imagen digital de la Historia, Kirsch, Russell A., "Russell A. Kirsch", NISTS Museum.	7
1.8. Ejemplo de imagen Transmitida en semitonos desde la Luna por la misión Apolo. a) Antes del procesamiento en semitonos y b) después del procesamiento que suaviza los tonos a escala de grises.	7
1.9. Primera cámara digital Steve Sasson quien trabajaba para la Kodak.	8
2.1. Espectro visible de luz para el ser humano	10
2.2. El Ojo humano y sus partes básicas.	10
2.3. Construcción geométrica de la imagen mediante cámara estenopeica. En la imagen, se representa una aproximación a la vista lateral.	12
2.4. Construcción geométrica de la imagen mediante una lente delgada convergente.	14
2.5. La cámara digital y su partes básicas.	17
2.6. Principio de funcionamiento de una cámara digital.	17
2.7. Representación de las celda fotosensibles.	18
2.8. Artificio para tomar fotografías a color con una cámara monocromática. Para esto se usan filtros de colores y se guarda la secuencia de las tres fotografías roja, verde y azul.	18
2.9. Arreglo para tomar fotografías a color con tres arreglos de sensores usando prismas traslúcidos.	19
2.10. Principio de funcionamiento de una cámara digital a color con filtro Bayer.	19
2.11. Cámara digital a color con filtro Bayer antes del algoritmo de interpolación y después de este.	20
2.12. Principio de cámara oscura para formación de imágenes en escala de grises y a color con filtro Bayer.	20
2.13. imagen de Cartagena descompuesta en cada una de sus capas del color en formato Raw del filtro Bayer (primera fila) y su versión interpolada y a color de estas (segunda fila).	21
2.14. Comparación entre una imagen tomada con una cámara de color tipo Bayer de 5 megapíxeles (1CCD, izquierda) y una cámara JAI AT-200 de 2 megapíxeles con sensor 3CCD (derecha). A pesar de que la cámara Bayer tiene mayor resolución, presenta contaminación de color, menor diferenciación entre colores similares y menor nitidez, debido al uso de filtros de polímero y el proceso de interpolación.	21
2.15. Resultados de la ejecución del código del ejemplo con las imágenes en escala de grises (a la izquierda) y a color (a la derecha) del ejemplo en Python.	22
3.1. Ejemplos de imágenes. a) Escultura localizada en el Museo Metropolitano de Arte de Nueva York. b) Pintura encontrada en en Museo del Vaticano. c) Fotografía tomada a la entrada del Museo del Louvre en París. Fuente: Los autores.	25
3.2. a) Muestra de píxeles representados en escala de grises. b) Escala de grises de 0 a 255.	26
3.3. Ejemplos de imagen digital. a) imagen digital con alta detalle. b) imagen digital con aproximación donde se visualizan cada uno de los píxeles. c) Algunos píxeles de la aproximación con su valor de las componentes R (rojo),G (verde) y B (azul).	26

3.4. Descomposición por pixeles de una imagen a color. a) Imagen a color. b) Pixel de una imagen a color con tres intensidades. c) Pixel de una capa de la imagen a color con una sola intensidad.	27
3.5. Resultado de la ejecución de código de Python para generar colores en una imagen a partir del modelo RGB. En ella se aprecian los colores rojo, verde, azul, cian, amarillo, magenta, negro, blanco y gris.	28
3.6. Imagen a diferentes resoluciones espaciales	28
3.7. Escala de tonalidades con diferentes números de bits.	29
3.8. Imagen a diferentes profundidades de bits del color.	29
3.9. Detalle que evidencia los efectos de cada formato sobre una imagen. a) Imagen en formato JPG con artefactos. b) Imagen en formato PNG. c)Imagen en formato GIF con reducción del color. d) Imagen en formato TIF. e) Imagen en formato BMP sin compresión. f)Imagen en formato RAW sin reconstrucción.	30
 4.1. Ejemplo de histograma de frecuencias.	36
4.2. Histogramas de dos imágenes diferentes en escala de grises que poseen igual histograma.	37
4.3. Imagen en escala de grises de 8 bits por color y su respectivo histograma para cada capa del modelo RGB.	37
4.4. Ejemplos de análisis del histograma para evaluar brillo y contraste.	39
4.5. Separación de los canales del color RGB de la imagen y sus respectivos histogramas en Python.	40
4.6. Ejemplos de imágenes con diferentes brillos. a) Bajo brillo, b) Alto brillo.	40
4.7. Ejemplos de imágenes con diferentes contrastes. a) Alto constaste, b) Mediano constraste y c) Bajo contraste.	41
4.8. Dos figuras A y B sobre una amplia escala de grises. Aunque no resulta explícito, los tonos de grises son iguales en ambos lados.	41
4.9. Histogramas para la imagen con $K = 2^3 = 8$ posibles valores de intensidad.	43
4.10. a) Histograma de entrada y su correspondiente. b) Histograma de salida deseado. c)Histogramas encarados	45
4.11. Comportamiento del gamma. a) Gamma lineal. b) Gamma mayor que uno. c) Gamma menor que uno y mayor que cero.	46
4.12. Operación de ajuste de contraste.	47
4.13. Imagen de Cartagena con diferentes ajustes de Gamma. En la parte superior su resultado y en la inferior su histograma. a) y b) con $\text{Gamma}=0.5$, c) y d) con $\text{Gamma}=1$, y e) y f) con $\text{Gamma}=1.5$	48
4.14. Imagen de 3 bits.	48
4.15. Histograma de frecuencias.	49
4.16. Imagen de 3 bits ajustada.	50
4.17. Histograma de frecuencias ajustado.	50
4.18. Ajuste de la imagen de 'cartagena.jpg' usando función exponencial. (a) Imagen ajustada y (b) Histograma ajustado.	51
4.19. Ajuste de la imagen de 'cartagena.jpg' usando función logaritmo. (a) Imagen ajustada y (b) Histograma ajustado.	51
4.20. Criterios de selección de los valores de E_m y E_M . En (a) se muestra el histograma de probabilidad con los límites y en (b) se aprecian las áreas bajo la curva del histograma acumulado.	52
4.21. La parte superior ilustra los histogramas de las fotografías tomadas en escala de grises, el la central se aprecia una vez se han ajustado las intensidades y, en la parte inferior de izquierda a derecha, la imagen de referencia, la compuesta sin ajustar y, finalmente, la ajustada.	55
4.22. Modelo para la ecualización del histograma en su versión discreta. (a) Histograma de entrada de probabilidad y (b) Histograma ecualizado deseado de salida.	56
4.23. Imagen de entrada 3 bits.	57
4.24. Histograma de frecuencias h	58
4.25. Imagen ecualizada de 3 bits.	58
4.26. Histograma ecualizado de 3 bits.	59
4.27. La parte superior ilustra los histogramas de las fotografías tomadas en escala de grises, el la central se aprecia una vez se han ecualizado las intensidades y, en la parte inferior de izquierda a derecha, la imagen de referencia, la compuesta sin ajustar y, finalmente, la ecualizada.	62
4.28. Transformación del histograma por especificación.	65
4.29. Imágenes de entrada E y referencia R de 3 bits y 4×4 para el ajuste por especificación.	65
4.30. a) Histograma Imagen de referencia R vs b) Histograma de Imagen de entrada E.	66

4.31. Correspondencia entre histogramas acumulados de la imagen de entrada <i>E</i> y la <i>y</i> imagen de referencia <i>R</i>	67
4.32. a) Histograma Imagen de referencia <i>R</i> vs b) Histograma de Salida <i>S</i>	68
4.33. Histograma de Referencia vs Imagen de Entrada.	69
4.34. Histograma de Salida vs Histograma de Entrada.	69
4.35. Histograma por especificación entre dos imágenes, permitiendo que la distribución de niveles de intensidad de una imagen de entrada se asemeje al histograma de una imagen de referencia.	71
 5.1. Combinación aditiva del modelo básico rojo, verde y azul como fuente para generar más colores.	76
5.2. Experimento CIE RGB. (a) Valores de estímulos RGB positivos. (b) Valores de estímulos RGB negativos.	77
5.3. Curvas de los modelos (a) CIE RGB y (b) CIE XYZ.	77
5.4. (a) Modelo del color XYZ y (b) Diagrama cromático XYZ.	78
5.5. Representación del espacio de color RGB como un cubo unitario tridimensional.	79
5.6. Imagen a color y sus correspondientes canales RGB.	79
5.7. Imagen a color y sus correspondientes canales CMY.	80
5.8. Espacio de color HSV.	81
5.9. Imagen a color y sus correspondientes canales HSV.	81
5.10. Segmentación del color amarillo de la bandera de Colombia en la imagen de Cartagena usando RGB.	96
5.11. Segmentación del color amarillo de la bandera de Colombia en la imagen de Cartagena usando HSV.	97
5.12. Segmentación del color amarillo de la bandera de Colombia en la imagen de Cartagena usando CIELAB.	97
5.13. Segmentación de la bandera de Colombia en su franja amarilla usando el modelo RGB. En la parte inferior, se muestran la imagen a color, luego la máscara de segmentación y, finalmente, la región amarilla segmentada.	100
5.14. Segmentación de la bandera de Colombia en su franja amarilla usando el modelo HSV. En la parte inferior, se muestran la imagen a color, luego la máscara de segmentación y, finalmente, la región amarilla segmentada.	102
5.15. Segmentación de la bandera de Colombia en su franja amarilla usando el modelo CieLab. En la parte inferior, se muestran la imagen a color, luego la máscara de segmentación y, finalmente, la región amarilla segmentada.	105
 6.1. Descomposición de las campas del modelo RGB de la imagen de la bandera.	110
6.2. Imagen a color y su respectiva representación en escala de grises mediante le promedio simple.	110
6.3. Imagen a color y su respectiva representación en escala de grises usando la luminancia.	111
6.4. Imagen a color y su respectiva representación en escala de grises usando la luminancia. La recomendación NTSC 601 es la más utilizada.	112
6.5. Conversión directa de gris a color con base en NTSC 601.	113
6.6. Proceso de segmentación por colo de la imagen de un auto amarillo usando le modelo del color HSV y la escala de grises.	116
 7.2.	120
7.1. Efecto del umbral sobre el histograma. a) Binarización y b) Umbralización.	120
7.3. Histograma para cada una de las capas de color RGB.	121
7.4. (a) Imagen original de un mango con fondo uniforme binarizada con los métodos globales (b) Iterativo de Ridler, (c) Entropía y (d) Otsu.	122
7.5. Imagen de 3 bits a binarizar.	125
7.6. Imagen umbralizada por el método Ridler para valor $T = 3$	128
7.7. Imagen umbralizada por el método Entropía para valor $T = 4$	130
7.8. Histograma inicial para la imagen de 3 bits.	130
7.9. Representación del umbral $T = 0$	131
7.10. Representación del umbral $T = 1$	131
7.11. Representación del umbral $T = 2$	132
7.12. Representación del umbral $T = 3$	132
7.13. Representación del umbral $T = 4$	133
7.14. Representación del umbral $T = 5$	133
7.15. Representación del umbral $T = 6$	134
7.16. Representación del umbral $T = 7$	134
7.17. Imagen umbralizada por el método OTSU para valor $T = 3$	135

7.18. Se muestra el proceso detallado de segmentación de un mango a partir de sus componentes de color Rojo (R), Verde (G) y Azul (B). Este se lleva a cabo mediante la aplicación del algoritmo de Otsu, que permite determinar un umbral óptimo para la binarización, separando el objeto de interés (el mango) del fondo. Para este caso se usó el generado por el color azul.	137
7.19. (a) Imagen original de un documento con sombras, b) binarizada con Ridler, (c) binarizada con Otsu y (d) binarizada con Entropía.	138
7.20. (a) Imagen original de un documento con sombras y (b) la versión binarizada con Bradley.	141
7.21. Binarización de una imagen de 3×3 y 3 bits mediante el método de Niblack. Se amplía la matriz I, se calculan los promedios y desviaciones estándar locales, y se aplica la fórmula de Niblack para obtener la matriz de umbrales T. Finalmente, se presenta la imagen adaptativamente binarizada.	143
7.22. Binarización de una imagen de 3×3 y 3 bits mediante el método de Sauvola. Se amplía la matriz I, se calculan los promedios y desviaciones estándar locales, y se aplica la fórmula de Sauvola para obtener la matriz de umbrales T. Finalmente, se presenta la imagen adaptativamente binarizada.	144
7.23. Binarización de una imagen de 3×3 y 3 bits mediante el método de Bradley. Se amplía la matriz I, se calculan los promedios y desviaciones estándar locales, y se aplica la fórmula de Bradley para obtener la matriz de umbrales T. Finalmente, se presenta la imagen adaptativamente binarizada.	145
7.24. La Figura muestra la segmentación de una imagen en escala de grises utilizando el umbral global de Otsu y el método adaptativo de Bradley. Se observa cómo Otsu aplica un único umbral, mientras que Bradley ajusta localmente el umbral según la iluminación.	147
 8.1. (a) Imagen con Ruido Sal y Pimienta y (b) corte de la misma fila sin ruido (azul) y con ruido Sal y Pimienta (punto rojo). Nótese los saltos entre 0 y 255 en las intensidades cuando hay ruido.	150
8.2. (a) Imagen con Ruido Gaussiano y (b) corte de la misma fila sin ruido (azul) y con ruido Gaussiano (punto rojo). Nótese que las intensidades se ven afectadas de manera regular sin importar si su valor es alto o bajo. El ruido no depende de la intensidad del pixel.	151
8.3. Función de distribución Gaussiana.	152
8.4. (a) Imagen con Ruido Speckle y (b) corte de la misma fila sin ruido (azul) y con ruido Speckle (punto rojo). Nótese que las intensidades se ven afectadas de forma irregular dependiendo de su valor. Para intensidades bajas, el ruido es poco y para intensidades altas el efecto de este es mayor. El ruido depende de la intensidad del pixel.	153
 9.1. Principio conceptual de la correlación en imágenes: En el proceso consiste en desplazar el kernel K sobre la imagen $I(x, y)$. Para cada posición del kernel, se realizan multiplicaciones elemento a elemento entre los valores del kernel y los correspondientes de la imagen bajo el kernel. Posteriormente, se suman todos los productos obtenidos. El valor resultante se asigna a la posición $I'(x, y)$ en la imagen de salida. Esto se repite a medida que el kernel se mueve por toda la imagen, produciendo una nueva filtrada.	164
9.2. Funcionamiento principal del filtro. La imagen I se convierte en I' una vez se ha aplicado el filtro promedio artimético.	170
9.3. Funcionamiento principal del filtro mínimo con ventana de 3×3 . (a) La imagen I solo con ruido Sal y Pimienta al 0.01. (b) La imagen I' con filtrado mínimo. Nótese que se han enfatizado la Pimienta (Puntos negros).	172
9.4. Funcionamiento principal del filtro máximo con ventana de 3×3 . (a) La imagen I solo con ruido Sal y Pimienta al 0.01. (b) La imagen I' con filtrado máximo. Nótese que se han enfatizado la Sal (Puntos blancos).	172
9.5. Funcionamiento principal del filtro de rango con ventana de 3×3 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado de rango. Es detecta regiones donde hay cambios.	173
9.6. Funcionamiento principal del filtro de punto medio con ventana de 3×3 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado de punto medio.	173
9.7. Ejemplo de cálculo de un filtro de mediana para una ventana de 3×3 pixeles.	174
9.8. Funcionamiento principal del filtro mediana con ventana de 3×3 . (a) La imagen I solo con ruido Sal y Pimienta al 0.01. (b) La imagen I' con filtrado mediana. Nótese que bastante efectivo en la eliminación de los puntos Sal y Pimienta (blancos y negros).	175
9.9. Funcionamiento principal del filtro moda con ventana de 11×11 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado moda. Como se puede apreciar, esta tiene un aspecto artístico de pintura al óleo difuminado.	176

9.10. Funcionamiento principal del filtro desviación estándar con ventana de 3×3 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado desviación estándar. Resulta ser adecuado para detectar regiones que cambian, por lo que en este caso, enfatiza los bordes.	176
9.11. Funcionamiento principal del filtro entropía con ventana de 3×3 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado entropía. Resulta ser adecuado identificar texturas en el reconocimiento de patrones.	177
9.12. Filtros de procesamiento digital aplicados a la Catedral Primada de Bogotá. (a) Imagen original; (b) Orden mínimo que resalta zonas oscuras; (c) Orden máximo que enfatiza áreas brillantes; (d) Mediana que preserva bordes mientras reduce ruido; (e) Moda que destaca valores predominantes; (f) Rango que muestra diferencias locales de intensidad; (g) Desviación estándar que visualiza variabilidad local; (h) Entropía que identifica regiones con mayor complejidad informativa.	179
9.13. Funcionamiento principal del filtro Gaussiano con ventana de 3×3 y desviación de 0.5. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Gaussiano. Esta suaviza la imagen y se ajusta más al modelo de desenfoque por plano de formación en imágenes que el promedio.	180
9.14. Visualización 3D del kernel Gaussiano de 31×31 y con desviación estándar de $\sigma = 5$	181
9.15. Se presentan tres imágenes: (a) la imagen original, (b) el resultado del filtrado con un filtro de promedio y (c) el resultado del filtrado con un filtro gaussiano.	183
9.16. Imagen primera derivada de una transición de blanco a negro y de negro a blanco.	185
9.17. Ejemplos de bordes y su respectiva función. Se muestra una imagen con un cambio brusco de intensidad, otra con un degradado gradual y otra con dos zonas de degradado.	185
9.18. Dirección del gradiente, magnitud y sentido. Se aprecian uno horizontal, vertical y diagonal.	186
9.19. Funcionamiento principal del filtro Laplaciano de 4 puntos con ventana de 3×3 y $\alpha = 0$. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Laplaciano de 4. Es adecuada para encontrar border en las imágenes, aun sueve usarse en ocmpaía de otros filtros como el Gaussiano ya que enfatiza el ruido.	189
9.20. Funcionamiento principal del filtro Laplaciano de 4 puntos con ventana de 3×3 y $\alpha = 0$. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Laplaciano de 4. Es adecuada para encontrar bordes en las imágenes, al usarse acompañado de otros filtros como el Gaussiano, ya que por sí solo que enfatiza el ruido.	190
9.21. Kernel LoG de 31×31 y con $\sigma = 5$	193
9.22. Funcionamiento principal del filtro Laplaciano del Gaussiano con ventana de 7×7 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Laplaciano del Gaussiano. Es adecuado para encontrar el borde en las imágenes, al ya que suaviza el ruido antes de buscar los contornos.	194
9.23. Funcionamiento principal del filtro Sharpen (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Sharp. Se usa para resaltar los bordes de los objetos en imágenes con falta de detalles.	195
9.24. Funcionamiento principal del filtro Sobel en X. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Sobel en la dirección X. Es adecuado para encontrar líneas horizontales y verticales en las imágenes.	197
9.25. Funcionamiento principal del filtro Sobel en Y. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Sobel en la dirección Y. Es adecuado para encontrar líneas horizontales y verticales en las imágenes.	197
9.26. Funcionamiento principal del filtro Prewitt en X. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Prewitt en la dirección X. Es adecuado para encontrar líneas horizontales y verticales en las imágenes.	198
9.27. Funcionamiento principal del filtro Prewitt en Y. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Prewitt en la dirección Y. Es adecuado para encontrar líneas horizontales y verticales en las imágenes.	199
9.28. Comparación de filtros pasa altos aplicados a una imagen de Bogotá. (a) Imagen original; (b-c) Filtros Prewitt en direcciones X e Y que detectan bordes verticales y horizontales; (d-e) Filtros Sobel en direcciones X e Y con mayor robustez ante ruido; (f) Filtro Laplaciano del Gaussiano (LoG) que resalta contornos en todas las direcciones.	200
10.1. Se aprecian algunas transformaciones geométricas afines y proyectivas, incluyendo la translación, el escalamiento, la rotación, el cizallamiento y la transformación proyectiva de cuatro puntos.	204
10.2. Transformación directa para determinar la nueva posición en las transformaciones geométricas. En este caso se recorre la imagen fuente I en la búsqueda de la posición donde se reubicará la intensidad en la imagen destino I'	205

10.3. Transformación inversa para determinar la nueva posición en las transformaciones geométricas. En este caso se recorre la imagen destino I' en la búsqueda de la posición en la imagen fuente I donde está la intensidad que le corresponde.	206
10.4. Representación de la translación de una forma (línea continua) a una posición final (línea discontinua) dentro de una escena. Nótese que, aunque los puntos cambian de posición, se mantiene el paralelismo final.	208
10.5. A la izquierda: imagen original tomada en el Museo Botero de la ciudad de Bogotá de sus obras "La Carta" y "Naranjas". A la derecha: ejemplo de imagen transladada.	209
10.6. Representación del escalamiento de una forma (línea continua) a una tamaño final (línea discontinua) dentro de una escena. Nótese que, aunque las líneas cambian de longitud, se mantiene el paralelismo final.	213
10.7. A la izquierda: imagen original tomada en el Museo Botero de la ciudad de Bogotá de sus obras "La Carta" y "Naranjas". A la derecha: ejemplo de imagen escalada.	214
10.8. Representación de la rotación de una forma (línea continua) a una posición final (línea discontinua) al rededor de un punto dentro de una escena. Nótese que, las líneas no cambian de longitud y se mantiene el paralelismo final.	218
10.9. Representación geométrica para la obtención de los puntos de giro.	219
10.10A la izquierda: imagen original tomada en el Museo Botero de la ciudad de Bogotá de sus obras "La Carta" y "Naranjas". A la derecha: ejemplo de imagen rotada.	220
10.11Transformaciones aplicadas a una imagen original. (a) Rotación de 60 grados, donde la imagen se gira manteniendo su estructura. (b) Escalado, reduciendo el tamaño de la imagen mediante un factor determinado. (c) Recorte de una región específica de la imagen original, seleccionando un área de interés. (d) Traslación de la imagen recortada a una nueva posición dentro del mismo lienzo.	226
10.12Representación del Cizallamiento de una forma (línea continua) a una forma final (línea discontinua) dentro de una escena. Nótese que, aunque las líneas cambian de ángulo, se mantiene el paralelismo final.	227
10.13A la izquierda: imagen original tomada en el Museo Botero de la ciudad de Bogotá de sus obras "La Carta" y "Naranjas". A la derecha: ejemplo de imagen Cizallada.	228
10.14Representación de la proyección de una forma (línea continua) a una posición final (línea discontinua) dentro de una escena. Nótese que, algunas líneas cambiar de longitud y otras pierden el paralelismo final.	231
10.15En la (a), se presenta la imagen de entrada, donde se han definido un conjunto de puntos de referencia iniciales. En (b), se observa la imagen transformada, obtenida mediante la matriz proyectiva H . La imagen original fue tomada en el Museo Botero de la ciudad de Bogotá de sus obras "La Carta" y "Naranjas".	235
10.16Se muestra de manera conceptual el proceso de interpolación por el Vecino más cercano. Nótese que los valores se repiten y pueden dar bordes dentados.	235
10.17Interpolación bilineal. Estar se acerca al centro a tra'ves de inicialmente hacer interpolaciones lineales simples.	236
10.18Primera aproximación lineal para I_{y_1}	236
10.19Segunda aproximación lineal para I_{y_2}	237
10.20Interpolación lineal de los casos anteriores, conducente a la interpolación bilineal para I	238
10.21Intensidad del punto $I(12, 2)$	239
 11.1. Explicación conceptual de algunas operaciones de conjuntos y lógicas básicas. (a) Unión y OR lógico, (b) Intersección y AND lógico y (c) Complemento y NOT lógico.	244
11.2. Ejemplos de Elementos Estructurante. a) Cuadrado, b) Rectangular, c) Disco, d) Cruz, e) Lineal y f) Diamante.	246
11.3. Proceso de Erosión morfológica. a) Imagen en escala de grises, b) Imagen Binarizada y c) Imagen Erosionada.	248
11.4. Ejemplo de Erosión. (a) para un pixel y (b) paara todo los pixeles.	249
11.5. Proceso de Dilatación morfológica. a) Imagen en escala de grises, b) Imagen Binarizada y c) Imagen Dilatada.	250
11.6. Ejemplo de Dilatación. (a) para un pixel y (b) para todo los pixeles.	251
11.7. Proceso de Apertura morfológica. a) Imagen en escala de grises, b) Imagen Binarizada y c) Imagen con Apertura.	253

11.8. Ejemplo de apertura. Ejemplo de Cerramiento. (a) Primero se realiza la erosión y (b) seguidamente la dilatación con el mismo elemento estructurante	254
11.9. Proceso de Cerramiento morfológico. a) Imagen en escala de grises, b) Imagen Binarizada y c) Imagen con Cerramiento.	255
11.10 Ejemplo de Cerramiento. (a) Primero se realiza la dilatación y (b) seguidamente la erosión con el mismo elemento estructurante.	256
11.11 Proceso de segmentación por cierre morfológico: a) imagen original, b) escala de grises, c) binarización, d) dilatación, e) cierre (dilatación seguida de erosión), f) segmentación de herramientas aplicando la máscara cerrada sobre la imagen original.	258
11.12 Plantilla para realizar la Erosión del ejercicio 1. Para efectos de facilidad en la visualización, se han dejado los 1 de color negro y los blancos como 0 en la gráfica.	260
11.13 Plantilla para realizar la Dilatación del ejercicio 2. Para efectos de facilidad en la visualización, se han dejado los 1 de color negro y los blancos como 0 en la gráfica.	260
11.14 Imagen circular para ejercicio de referencia de detección de bordes.	261
11.15 Plantilla general para realizar ejercicios de operaciones morfológicas.	262
 12.1. Conectores bidimensionales.	264
12.2. Proceso de etiquetado para una imagen binaria de 6x6.	265
12.3. Ejemplo del proceso de etiquetado de componentes conectados en una imagen binaria utilizando conectividad 4. Se muestra la asignación inicial de etiquetas, la detección de colisiones y la unificación final de etiquetas equivalentes.	267
12.4. Resultados del etiquetado de componentes conectados.(a) Imagen con las etiquetas mostradas en escala de grises.(b) Visualización de los componentes conectados en falso color RGB para facilitar su identificación.(c) Máscara aplicada sobre la imagen original a color, resaltando únicamente el componente seleccionado.	268

Introducción

En el panorama educativo actual, existen diversos libros de texto dedicados al Procesamiento de Imágenes y a la implementación de sus algoritmos. No obstante, pocos de ellos abordan estos temas desglosando las matemáticas subyacentes de manera que resulten accesibles para estudiantes con distintos perfiles profesionales. Este libro busca cerrar esa brecha, presentando ejemplos fundamentados en matemáticas discretas que ilustran paso a paso el funcionamiento de los algoritmos, permitiendo al lector implementarlos en cualquier lenguaje de programación.

El propósito principal de esta obra es proporcionar una comprensión profunda de los principios fundamentales que rigen el comportamiento de los algoritmos de Procesamiento de Imágenes, priorizando la comprensión conceptual por encima de la optimización orientada a producción. A través de ejemplos detallados, se pretende que los estudiantes comprendan los fundamentos teóricos y a su vez desarrollen habilidades prácticas para codificar los algoritmos por sí mismos.

El enfoque pedagógico está centrado en la aplicación práctica de los conceptos mediante ejercicios numéricos y ejemplos desarrollados en **Python** sin el uso de las librerías específicas tales como o aplicar librerías especializadas como **Matlab(Matworks)**, **scikit-learn** u **OpenCV**. no obstante lo autores reconocen que en términos de producción, ellas son el camino en la mayoría de los casos. La principal contribución de este material reside en su capacidad para presentar temas complejos de forma clara y accesible, partiendo de conocimientos matemáticos básicos adquiridos en los primeros años de formación en ingeniería o licenciatura. De este modo, se brinda al lector una base sólida para avanzar hacia estudios más profundos y aplicaciones más robustas en el campo del Procesamiento de Imágenes.

Es de destacar que las funciones implementadas en **Python** en este texto, se han inspirado en su sintaxis de **Matlab(Matworks)**, puesto que resulta ser un referente en el procesamiento de imágenes en entornos académicos y científicos.

Capítulo 1

Antecedentes

1.1. Antecedentes del Procesamiento de imágenes

El procesamiento de imágenes (conocido como IP por sus siglas en inglés, Image Processing) abarca un conjunto de técnicas que emplean algoritmos informáticos para analizar, mejorar, comprimir y reconstruir imágenes. Este campo encuentra una amplia variedad de aplicaciones en diversas áreas, como la astronomía, la medicina, la robótica industrial y las ciencias forenses, entre otras.

En la década de 1960, con el advenimiento de la informática y los avances en la tecnología de los sensores de imagen, comenzó a surgir el procesamiento digital de imágenes. En 1969, Willard S. Boyle (Amherst, Canadá, 19 de agosto de 1924 - 7 de mayo de 2011) y George E. Smith (White Plains, Nueva York, 10 de mayo de 1930) inventaron el primer dispositivo de carga acoplada CCD (Charge-Coupled Device, en inglés) el cual funciona con base en las teorías del efecto fotoeléctrico de Albert Einstein. Su principio básico transforma la luz en señales eléctricas, con lo que se logró la captura de imágenes a partir de arreglos. Esta invención les permitió ganar un Premio Nobel de Física en 2009. Actualmente, los CCD se utilizan en una amplia gama de aplicaciones, pero su principal uso es en la toma de imágenes digitales, lo que le ha llevado a ser la tecnología dominante para la captura en dispositivos electrónicos, puesto que ofrecen una alta calidad siendo a su vez muy eficientes. En la Figura 1.1 se aprecia una fotografía de la época del trabajo de los científicos inventores Willard S. Boyle y George E. Smith.

1.1.1. Algunos momentos Históricos de la Fotografía

El uso de la cámara ha permitido guardar momentos históricos y evidenciar los cambios en la humanidad reciente de una forma más objetiva. La revista TIME creó una lista de las 100 imágenes más influyentes jamás tomadas en su edición del 28 de noviembre de 2016 (<https://time.com/a4574500/most-influential-photos/>). Para hacerlo, unió a curadores, historiadores, editores de fotografía y fotógrafos famosos de todo el mundo. Según se indica, algunas imágenes están en la lista porque fueron las primeras de su tipo, otras porque dieron forma a la manera en que se piensa o simplemente cambiaron el estilo de la vida. Dentro de ellas este documento destaca:

En 1826, el francés Joseph Nicéphore Niépce diseñó una cámara oscura con la que proyectó escenas exteriores iluminadas por la luz solar desde su estudio. La escena fue proyectada en una placa de peltre (aleación de estaño, cobre, antimonio y plomo) tratada químicamente que, después de muchas horas, capturó una burda imagen de los edificios y los tejados. La fotografía mostrada en la Figura 1.2a es la primera imagen permanente conocida de la historia [38].

En 1895, Wilhelm Roentgen capturó los huesos de la mano de su esposa, Anna Bertha, con la primera



Figura 1.1: Inventores de la tecnología CCD. Willard S. Boyle (a la izquierda) y George E. Smith (a la derecha).

radiografía médica conocida. La imagen presentada en la Figura 1.2b revolucionó el diagnóstico y el tratamiento de lesiones y enfermedades que no eran evidentes a simple vista. Se menciona que Anna Bertha exclamó: “He visto mi muerte”; la primera vez que observó sus huesos. El descubrimiento de rayos X le valió a Wilhelm el primer Premio Nobel de física en 1901. No se puede dejar atrás, el valor de las imágenes diagnosticas, un aporte significativo que cada día salva más vidas alrededor del mundo [45].

El 6 de mayo de 1937, el periodista Sam Shere fotografió el siniestro en la estación aérea naval de Lakehurst, en New Jersey, Estados Unidos. Esta imagen, mostrada en la Figura 1.2c, se considera la más famosa del accidente, después de que llegó a la portada del primer álbum del grupo musical Led Zeppelin el 2 de enero de 1969 [51].

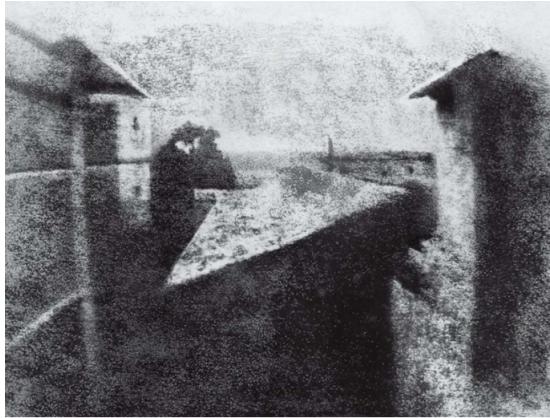
20 de julio de 1969, la imagen del segundo astronauta en pisar la luna, Buzz Aldrin, logró una posición destacada en la historia ya que mostraba, por primera vez, un hombre en la luna como se ve en la Figura 1.2d. Si bien Neil Alden Armstrong fue el primero en pisarla, este fue quien tomó también la fotografía de su inmortalizado compañero Aldin [34].

El 1 de abril de 1995, el telescopio espacial Hubble, a bordo del transbordador espacial Atlantis, tomó la imagen ilustrada en la Figura 1.2e de la Nebulosa del Águila localizada a 6.500 años luz de la Tierra. La fotografía está formada por 32 imágenes diferentes de cuatro cámaras separadas. Esta imagen es tan definida que se conoce como Pilares de la Creación. La fotografía solo se logró después de algunos intentos fallidos debido a defectos en la lente de fabricación [35].

En 1997, Philippe Kahn, un empresario de software norteamericano, construyó un sistema para poder enviar una foto de su hija recién nacida a más de 2000 personas, entre amigos y familiares, en tiempo real. Para esto conectó una cámara digital al teléfono celular, y usó un programa que había escrito en su computadora portátil en el hospital mientras su esposa daba a luz. La imagen mostrada en la Figura 1.2f es considerada la primera imagen tomada y transmitida por un celular. Si bien su logro tecnológico es muy destacable, el aporte más significativo está en los cambios socioculturales que ha traído la inmediatez de compartir imágenes a través de los dispositivos móviles [22].

1.1.2. La primera fotografía a color de la Historia

En 1855, el reconocido físico y matemático escocés, James Clerk Maxwell presentó ante la Royal Society de Edimburgo su teoría tricromática del color, la cual fue publicada en detalle en 1857. Esta desafía los conceptos previos sobre



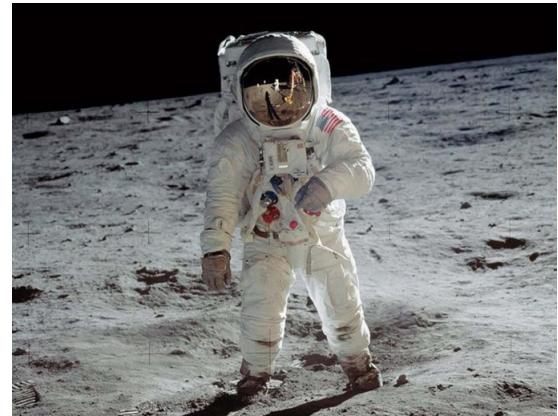
(a) Vista desde la ventana de Le Gras, 1826. Créditos: Joseph Nicéphore Niépce.



(b) La mano de la señora Wilhelm Röntgen, 1895. Créditos: Wilhelm Conrad Röntgen.



(c) El desastre de Hindenburg, 1937. Créditos: Sam Shere.



(d) Un hombre en la luna, 1969. Créditos: NASA.



(e) Pilares de la creación, 1995. Créditos: NASA.



(f) Primera foto con un teléfono celular, 1997. Créditos : Philippe Kahn.

Figura 1.2: La revista TIME creó una lista de las 100 imágenes más influyentes jamás tomadas.



(a) Cinta de tartán (Tartan Ribbon), Thomas Sutton, 1861.



(b) 1877, Louis Ducos du Hauron, View of Agen.

Figura 1.3

la naturaleza del color, y establecía que los colores visibles podían ser recreados mediante la combinación aditiva de mezclas de luz roja, verde y azul. El 17 de mayo de 1861, Maxwell llevó a la práctica su propuesta al realizar la primera demostración exitosa de fotografía a color durante una conferencia en la Royal Institution de Londres. Para hacerlo, utilizó tres transparencias separadas captadas de color rojo, verde y azul, las cuales proyectó de forma superpuesta con linternas de la época, logrando recrear una imagen en color que mostraba un tartán escocés sentando así las bases para la fotografía a color. La Figura 1.3a muestra una toma del registro denominada *Tartan Ribbon*, la cual si bien no representa los colores perfectamente, debido a las limitaciones de los materiales fotográficos de la época, le permitió a Maxwell ser denominado el padre de la fotografía a color [30,31].

En 1861, poco tiempo después de la demostración de Maxwell, el francés Louis Ducos Du Hauron, presentó una técnica para crear fotografías impresas a color usando pigmentos. Su método necesitaba también de tres negativos generados a través de filtros de color rojo, verde y azul. Las imágenes positivas producidas se teñían individualmente y luego se superponían para generar la fotografía a color impresa. Este método resulta ser la base práctica de los procesos de fotografía a color actuales. La más reconocida de sus fotografías a color es “Vista de Agen”, la cual corresponde a un paisaje en el sur de Francia tomada en 1877. Véase Figura 1.3b.

1.1.3. Retoque de imágenes

La manipulación de las imágenes parece un tema reciente. No obstante, esta emergió casi desde los orígenes de la fotografía, bien fuese por defectos en el proceso de revelado, deterioro por el paso del tiempo o, sorprendentemente, también asuntos caprichosamente personales. Esta labor no se realizaba mediante algún tipo de software, realmente era un trabajo artístico que exigía mucha destreza con el bisturí, el pegante, la pintura y el aerógrafo. Algunos de los casos históricamente destacados se produjeron en la administración de la ya desaparecida Unión soviética. Stalin solía ordenar remover de los registros fotográficos e históricos a “camaradas” que consideraba desleales a su causa [24,47].



Figura 1.4: Edición de imágenes por parte de la Unión Soviética. Imagen editada a la izquierda y original a la derecha.

Este es el caso de La fotografía ilustrada en la Figura 1.4. Ella es una de las innumerables imágenes donde se removió artísticamente a Leon Trotsky por orden de Stalin. El líder soviético, adicionalmente, exilió a Trotsky por crear una oposición fallida a su liderazgo, después de haberle apoyado en la fundación de su movimiento comunista.

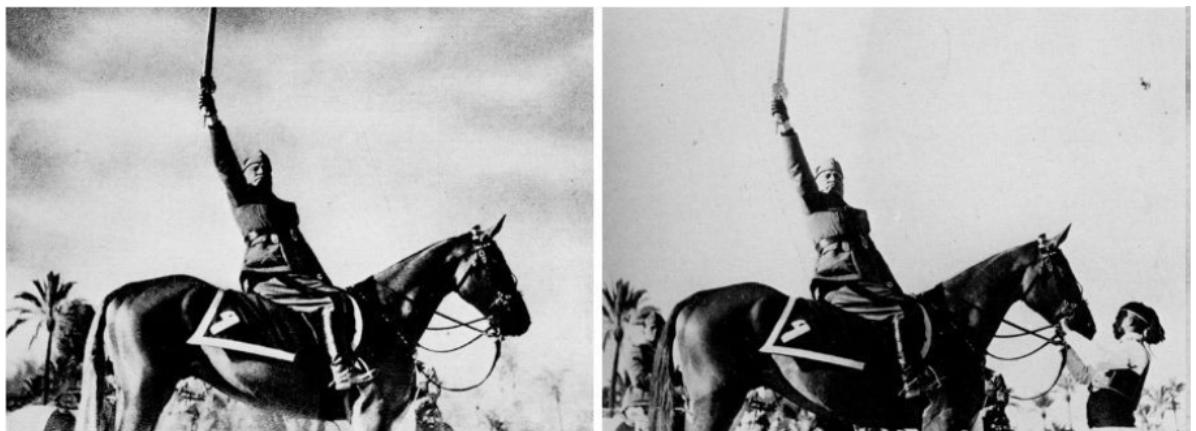


Figura 1.5: Edición de imágenes por parte de Benito Mussolini. Imagen editada a la izquierda y original a la derecha.

El político y líder italiano Benito Mussolini, creador del partido fascista, ordenó en 1942 remover la rienda del caballo de la fotografía con el objetivo de producir una imagen más heroica. Véase la Figura 1.5.



Figura 1.6: Edición de imágenes por parte de Adolf Hitler. Imagen editada a la izquierda y original a la derecha.

En las fotografías mostradas en Figura 1.6 sobresale la figura de Adolf Hitler en el centro. Joseph Goebbels, segundo hombre desde la derecha, fuese borrado de la original. Aún no se conoce la razón por la cual le eliminaron del retrato.

1.1.4. Sobre la Transmisión de imágenes

Por otro lado, en 1880, Shelford Bidwell notó que cuándo la luz brillaba sobre una lámina de selenio, su resistencia aumentaba. A partir de esto, concluyó que al ir realizando variaciones de luz podría generar una señal intermitente similar a un código Morse para transmitirla. Para probar su idea, construyó un telégrafo para fotos que recorría una imagen muy simple a tramos y creaba una copia sobre papel fotosensible a distancia. La primera imagen transmitida, utilizando estas tecnologías del siglo XIX, no es algo que se pueda encontrar o afirmar fácilmente. Esto se debe a que los primeros experimentos y desarrollos en la transmisión de imágenes por cable, como los realizados por Alexander Bain, Frederick Bakewell, y Giovanni Caselli, eran experimentales y a menudo se transmitían entre ubicaciones específicas para demostraciones o pruebas, que incluían textos manuscritos, firmas o dibujos simples. A manera de ejemplo, la fotografía mostrada en la Figura 1.7a es una de las primeras imágenes transmitidas por cable usando fototelegrafía que usaba papel fotosensible. El equipo construido por el Dr. Shelford Bidwell un año más tarde de su descubrimiento.

En 1957, Russell Kirsch trabajaba en la Oficina Nacional de Normas, lo que le permitió tener acceso una de las primeras computadoras programables en las cuales también había trabajado en su diseño. Esta se usaba realmente para cálculos de la guerra, pero se les permitía operarla para otro tipo de investigaciones. Kirsch y sus colegas rastrearon la imagen de su hijo recién nacido, usando un escáner de tambor también construido por ellos, el cual capturó la información en forma de pequeños puntos que se convirtieron en unos y ceros. La imagen se componía de 30.976 píxeles y tenía dimensiones de $5\text{cm} \times 5\text{cm}$. Esta se convirtió en la primera imagen digital de la historia que podía ser almacenada y reproducida en la pantalla de un osciloscopio para esta época. No obstante, solo muchos años después, cuando se mejoraron los sistemas de almacenamiento y compresión de imágenes, se dio uso real a su hallazgo [25]. Véase la Figura 1.7b.

La imagen en semitonos mostrada en la Figura 1.8 corresponde a una de las fotografías distribuidas durante las misiones Apollo a la Luna. En aquel entonces, era común que las tomas se reprodujeran con esta técnica para su publicación en periódicos y revistas, utilizando patrones de puntos para simular gradientes de tonalidad. Esto se debía a las restricciones de la tecnología de impresión en blanco y negro de la época y su facilidad de transmisión. Para superar estas limitaciones y mejorar la calidad de las imágenes, se empleaban técnicas de procesamiento de imágenes como la transformada de Fourier, que permitía atenuar el ruido periódico y mejorar la claridad visual, resultando en fotografías de mayor calidad que las típicamente impresas en medios [3].

La primera cámara electrónica fue inventada en 1975 por el ingeniero Steve Sasson quien trabajaba para la Kodak. La cámara, de 0.01 mega píxeles y 3.6kg , solo registraba imágenes a escala de grises que se guardaban en una cinta magnética (casete), común para esos días, y se reproducía en una pantalla de televisión. Para ese entonces, capturar una fotografía tomaba alrededor de 23 segundos, un retroceso aparente comparado con el tiempo que le llevaba hacerlo a una cámara analógica del momento que era de unas fracciones de segundo. Según se indica, la primera

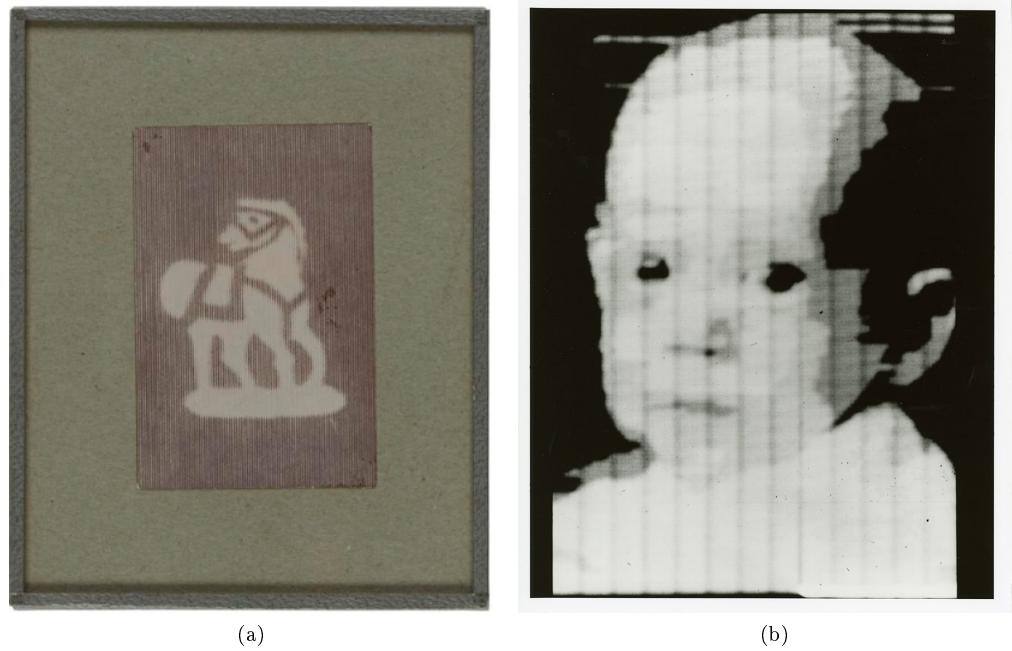


Figura 1.7: (a) Bidwell, Shelford, The Board of Trustees of the Science Museum, London, 1881 (b) Primera imagen digital de la Historia, Kirsch, Russell A., "Russell A. Kirsch", NISTS Museum.

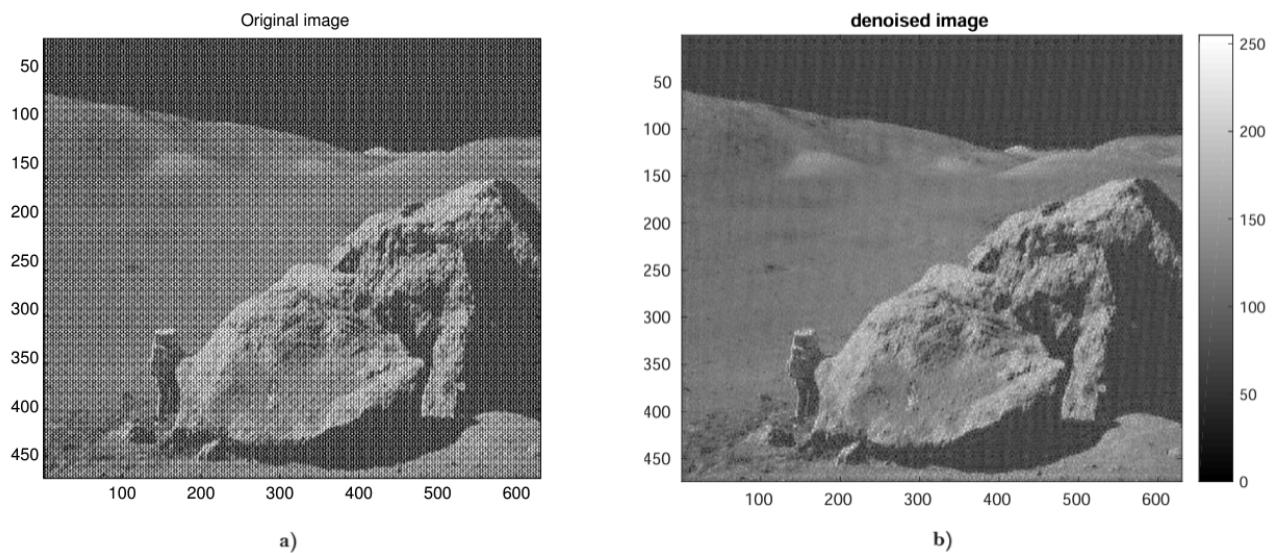


Figura 1.8: Ejemplo de imagen Transmitida en semitonos desde la Luna por la misión Apolo. a) Antes del procesamiento en semitonos y b) después del procesamiento que suaviza los tonos a escala de grises.

fotografía digital que se tomó fue de uno de los técnicos de laboratorio llamado Joy, pero desafortunadamente Steven no guardó la imagen [14,41]. En la Figura 1.9 se muestra el prototipo del cámara cargada en hombros por su creador y la pantalla de televisión donde se reproducía a partir de una cinta.



Figura 1.9: Primera cámara digital Steve Sasson quien trabajaba para la Kodak.

1.2. Preguntas

1.2.1. Preguntas de comprensión:

1. ¿Quién realizó la primera radiografía médica y qué mostró?
2. ¿Quiénes fueron los inventores de la tecnología CCD y en qué año la desarrollaron?
3. ¿Quién tomó la primera fotografía a color de la historia y cómo lo logró?
4. ¿Cuál fue la primera imagen digital de la historia y quién la creó?
5. ¿Qué astronauta aparece en la famosa fotografía del hombre en la luna?

1.2.2. Preguntas de aplicación y evaluación:

1. ¿Por qué la invención de la fotografía por Niépce fue tan revolucionaria e importante en la historia?
2. ¿Cómo ha cambiado la fotografía la manera en que registramos y compartimos momentos especiales?
3. ¿Por qué el desarrollo de la fotografía digital fue tan lento al inicio?
4. ¿Qué factores permitieron su expansión?
5. Evalúa el impacto ético de la manipulación de imágenes a lo largo de la historia.

Capítulo 2

La Cámara

2.1. La luz

La definición de luz (del latín lux, lucis) ha sido una de las más elaboradas a través de la historia de la ciencia. Muchas teorías se han planteado para explicarla, dentro de las cuales se encuentra aquella que describe a la luz como un haz de partículas radiantes llamadas fotones, los cuales son considerados como la mínima expresión de luz; en la actualidad se describe que los fotones se transmiten en un campo ondulatorio electromagnético. Por lo tanto se plantea que la luz posee un comportamiento dual, es onda y partícula a la vez; ya que los fenómenos de propagación se pueden explicar con la teoría ondulatoria y los fenómenos de interacción de la luz con la materia con el modelo corpuscular. En esta sección se describen brevemente algunas de las teorías que explican el comportamiento de la luz [39].

2.1.1. Naturaleza de la luz

Como ya se indicó, a la energía radiante se le asocia una naturaleza dual , la cual obedece a leyes que pueden explicarse, por ejemplo, usando un modelo de corriente de partículas o paquetes de energía (llamados fotones), o a partir de un tren de ondas transversales (Movimiento ondulatorio). En las ondas de luz, como en todas las ondas electromagnéticas, existen campos eléctricos y magnéticos en cada punto del espacio que fluctúan con rapidez. La luz visible es solo una pequeña parte del espectro electromagnético. En el espectro visible, las diferencias en longitud de onda se manifiestan como cambios de color. El rango visible va desde, aproximadamente, 380nm (nanómetros) del color violeta hasta unos 750nm del color rojo, ($1\text{mm} = 1000000\text{nm}$) aunque algunas personas pueden poseer un espectro más amplio. La luz blanca es una mezcla de todas las longitudes de onda visibles. La Figura 2.1 presenta el espectro de luz típico visible humano [49].

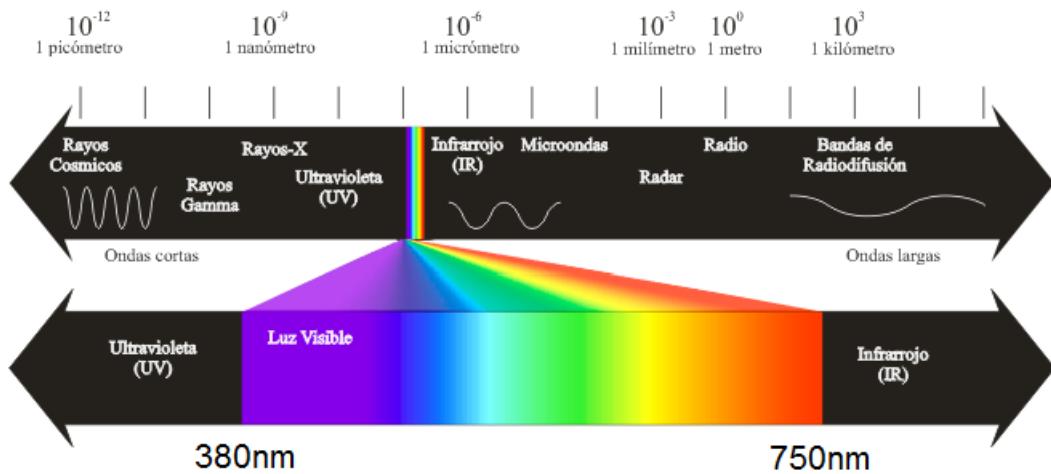


Figura 2.1: Espectro visible de luz para el ser humano

2.1.2. El ojo Humano

El ojo es un órgano que forma parte de uno de los sentidos más relevantes. Este percibe la luz, en el rango visible, y la lleva en forma de estímulos al cerebro que interpreta las formas, los colores y las dimensiones.

La córnea es la estructura externa transparente que se encuentra en la parte frontal del ojo rodeada por la esclerótica de color blanco. Esta ayuda a focalizar la luz que ingresa. Detrás de la córnea hay una membrana coloreada en forma de anillo llamada iris. Esta tiene una abertura circular ajustable denominada pupila, que puede expandirse o contraerse para controlar la cantidad de luz que ingresa al ojo.

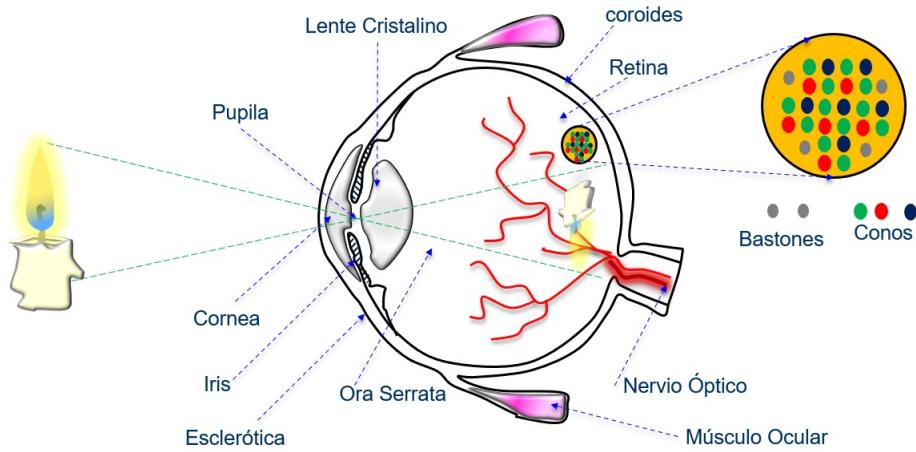


Figura 2.2: El Ojo humano y sus partes básicas.

De igual forma, detrás de la pupila se encuentra una estructura transparente e incolora denominada lente cristalina. Los músculos ciliares rodean la lente cristalina para mantenerla en su lugar, pero también juegan un papel importante en la visión. Cuando los músculos se relajan, tiran y apllanan la lente, permitiendo que el ojo vea objetos que están lejos y al contraerse hacen engrosar la lente para enfocar claramente objetos cercanos. El espacio entre la córnea y el iris está lleno de un líquido transparente denominado humor acuoso. La luz debe viajar a través de este humor antes de incidir sobre la capa sensible de células fotosensibles llamada retina. Ella es la más interna de las tres capas de tejido que forman el ojo. Sobre la retina hay millones de células sensibles a la luz denominadas bastones y conos. Los primeros se utilizan para la visión monocromática con poca luz, mientras que últimos se requieren para percibir el color y los detalles finos. Los conos son células especializadas para el color rojo, verde y azul, siendo su mayor porcentaje de verde. Cuando la luz golpea los bastones o los conos sobre la retina, se convierte en una señal

eléctrica que se transmite al cerebro a través del nervio óptico. Luego, el cerebro traduce las señales eléctricas en las imágenes. La capa media entre la retina y la esclerótica se llama coroides. La coroides contiene vasos sanguíneos que suministran a la retina nutrientes y oxígeno y eliminan sus productos de desecho. véase la Figura 2.2 para identificar sus partes [29].

2.2. La cámara

2.2.1. Formación del imagen

La formación de imágenes mediante sistemas ópticos simples, como la cámara estenopeica y la lente delgada, son la base fundamental para comprender cómo se proyecta el mundo tridimensional sobre un plano bidimensional. Estos permiten explicar, de manera geométrica cómo la luz transporta información visual desde una escena hasta un sensor. Su estudio introduce conceptos claves como la proyección, la escala y la inversión de imagen, a la vez que motiva la transición hacia modelos matemáticos discretizados que se usan en el procesamiento digital de imágenes.

2.2.2. Formación de imagen en la cámara estenopeica

Este tipo funciona con base en la proyección rectilínea de los rayos de luz a través de un pequeño orificio (pinhole), por lo que la formación de la imagen puede describirse utilizando semejanza de triángulos que es evidente en la Fig.2.3:

$$\frac{h'}{h} = \frac{q}{p} \quad (2.1)$$

donde:

- h : altura del objeto,
- h' : altura de la imagen proyectada,
- p : distancia del objeto al orificio,
- q : distancia del orificio al plano de imagen.

De esta relación se deduce:

$$h' = h \cdot \frac{q}{p}$$

La imagen resultante es invertida debido a la propagación recta de la luz, y su tamaño está directamente relacionado con la razón de distancias. El **aumento lateral** se expresa como:

$$m = \left| \frac{h'}{h} \right| = \frac{q}{p}$$

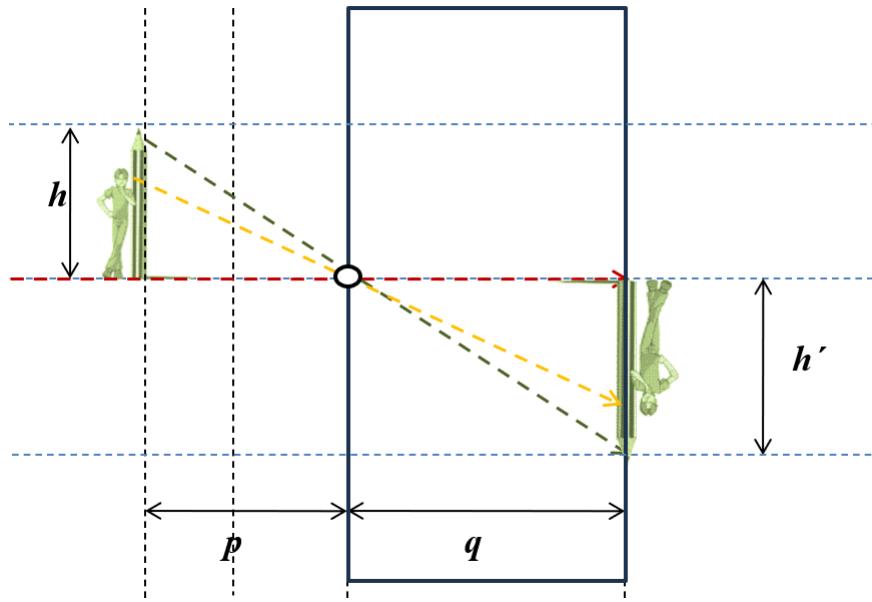


Figura 2.3: Construcción geométrica de la imagen mediante cámara estenopeica. En la imagen, se representa una aproximación a la vista lateral.

Este modelo es fundamental para entender los principios geométricos que subyacen a la captura de imágenes, como ocurre en los sensores digitales de las cámaras modernas.

Ejercicio: Formación de imagen en una cámara estenopeica

Se dispone de un objeto vertical de 10cm de altura, ubicado frente a una cámara estenopeica cúbica de 5 cm de profundidad, cuyo sensor cubre completamente la cara opuesta al orificio (de dimensiones 5 cm × 5 cm). Se desea que la imagen proyectada del objeto ocupe solamente la mitad de la altura del sensor.

Se plantea lo siguiente:

1. ¿A qué distancia del orificio debe colocarse el objeto para cumplir esta condición?
2. ¿Cuál es el valor del aumento lateral bajo estas condiciones?
3. Si el sensor está compuesto por un arreglo uniforme de sensores cuadrados de 50 μm de lado, sin espacio entre ellos, ¿cuántos sensores se requieren para cubrir completamente el plano del fondo de la cámara? ¿De qué resolución en píxeles se dice que es esta matriz?

Solución

1. Distancia necesaria para el encuadre deseado La formación de imagen en una cámara estenopeica puede modelarse mediante la semejanza de triángulos, expresada en la ((2.2)):

$$\frac{h'}{h} = \frac{q}{p} \quad (2.2)$$

donde:

- h : altura del objeto real,

- h' : altura de la imagen proyectada,
- q : distancia del orificio al plano del sensor,
- p : distancia del objeto al orificio.

Despejando p se obtiene (2.3):

$$p = h \cdot \frac{q}{h'} \quad (2.3)$$

Se desea que $h' = 2.5$ cm (la mitad del sensor), con $h = 10$ cm y $q = 5$ cm., por lo que substituyendo:

$$p = 10 \cdot \frac{5}{2.5} = 20 \text{ cm} \quad (2.4)$$

Por tanto, el objeto debe ubicarse a **20 cm del orificio** para que su imagen ocupe solamente la mitad de la altura del sensor.

2. Aumento lateral El aumento lateral se define como se muestra en 2.5:

$$m = \left| \frac{h'}{h} \right| = \frac{2.5}{10} = 0.25 \quad (2.5)$$

La imagen tiene un tamaño de una cuarta parte del objeto real.

2. sobre le sensor El área total a cubrir en el plano del sensor es de $5 \text{ cm} \times 5 \text{ cm}$, es decir, $50000 \mu\text{m} \times 50000 \mu\text{m}$. Si cada uno tiene un tamaño de $50 \mu\text{m} \times 50 \mu\text{m}$, el número de sensores por lado se calcula como:

$$\frac{50000 \mu\text{m}}{50 \mu\text{m}} = 1000 \text{ sensors por lado}$$

Por tanto, el número total de sensores (pixeles) necesarios para cubrir el fondo es:

$$1000 \times 1000 = 1000000.00 \text{ sensores (pixeles)}$$

La matriz se considera entonces de resolución 1000×1000 píxeles, lo que se conoce también como una imagen de 1 megapíxel.

2.2.3. Formación de imágenes con lentes delgadas

La formación de imágenes mediante **lentes delgadas convergentes (divergentes)** puede analizarse utilizando construcciones geométricas simples, como se muestra en Fig 2.4. En esta representación, basta con seguir el recorrido de tres rayos notables para determinar la ubicación y características de la imagen formada. La relación entre la distancia del objeto p , la distancia de la imagen q y la distancia focal f está dada por (2.6):

$$\frac{1}{p} + \frac{1}{q} = \frac{1}{f} \quad (2.6)$$

La distancia focal f está relacionada con el radio de curvatura R de la lente mediante:

$$f = \frac{R}{2} \quad (2.7)$$

El **aumento lateral** m , que indica el tamaño relativo de la imagen respecto al objeto, se calcula como:

$$m = \frac{h'}{h} = -\frac{q}{p} \quad (2.8)$$

La Ecuación (2.6) puede aplicarse tanto a lentes convergentes como divergentes si se adopta una convención de signos como la mostrada en la Tabla 2.1 .

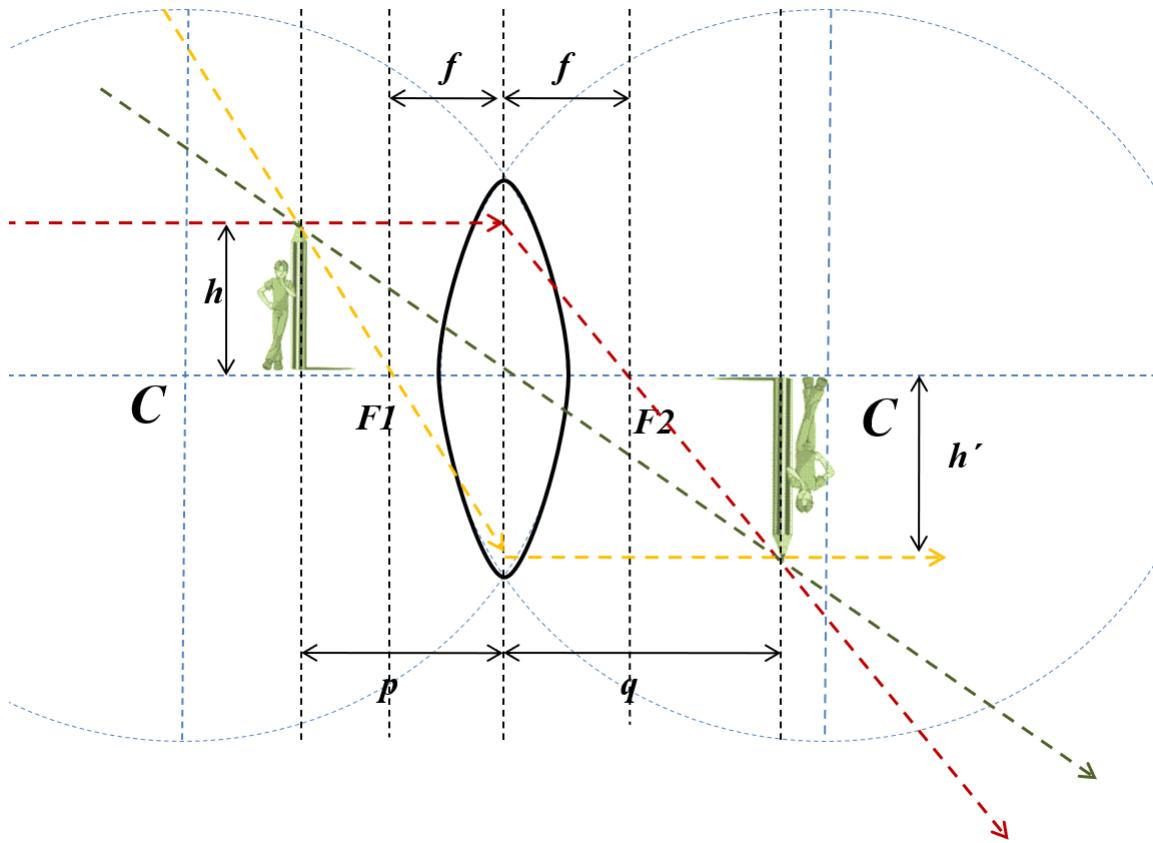


Figura 2.4: Construcción geométrica de la imagen mediante una lente delgada convergente.

La Tabla 2.1 resume los signos que se deben utilizar para cada elemento en el tratamiento de las imágenes formadas por lentes delgadas.

Elemento	Frente al Espejo	Detrás del Espejo	Cóncavo	Convexo	No invertida	Invertida
p	+	-				
q	+	-				
f			+	-		
R			+	-		
M				+	+	-

Cuadro 2.1: Resumen de signos para lentes delgadas.

Ejercicio: Formación de imagen mediante una lente convergente

Se dispone de un objeto vertical de 10 cm de altura, ubicado frente a una cámara equipada con una **lente delgada convergente de distancia focal $f = 4 \text{ cm}$** . El sensor se encuentra fijo a 5 cm detrás de la lente, ocupando completamente el fondo interno de la cámara, el cual tiene dimensiones de $5 \text{ cm} \times 5 \text{ cm}$. Se desea que la imagen proyectada del objeto **ocupe solamente la mitad de la altura del sensor**.

Se plantea lo siguiente:

1. ¿Cuál es el valor del aumento lateral bajo estas condiciones? ¿Qué indica sobre la orientación y escala de la imagen?
2. ¿A qué distancia de la lente debe colocarse el objeto para cumplir esta condición?
3. Compare las soluciones.

Solución**Datos conocidos:**

- Altura del objeto: $h = 10 \text{ cm}$
- Altura deseada de la imagen: $h' = 2.5 \text{ cm}$
- Distancia del sensor (plano imagen): $q = 5 \text{ cm}$
- Distancia focal de la lente: $f = 4 \text{ cm}$

1. **Aumento lateral** Usando los valores de $p = 20 \text{ cm}$ y $q = 5 \text{ cm}$:

$$m = -\frac{q}{p} = -\frac{5}{20} = -0.25 \quad (2.9)$$

El signo negativo indica que la imagen está invertida verticalmente respecto al objeto, y el valor absoluto indica que la imagen tiene un cuarto del tamaño original. Es decir, la imagen mide:

$$h' = m \cdot h = -0.25 \cdot 10 = -2.5 \text{ cm} \quad (2.10)$$

1. **distancia de la lente** Para una imagen real proyectada sobre el sensor, el aumento es negativo:

$$m = -\frac{q}{p} \quad (2.11)$$

Despejando p de ((2.11)):

$$p = -\frac{q}{m} = -\frac{5}{-0.25} = 20 \text{ cm} \quad (2.12)$$

Por tanto, el objeto debe colocarse a $p = 20 \text{ cm}$ frente a la lente para que su imagen proyectada ocupe la mitad del sensor.

3. Comparación con la cámara estenopeica En una cámara estenopeica, la relación de semejanza entre triángulos determina:

$$\frac{h'}{h} = \frac{q}{p} \quad (2.13)$$

Sustituyendo los valores dados:

$$\frac{2.5}{10} = \frac{5}{p} \Rightarrow p = \frac{5 \cdot 10}{2.5} = 20 \text{ cm} \quad (2.14)$$

En consecuencia, la distancia del objeto también debe ser de 20 cm en la cámara estenopeica para lograr una imagen de 2.5 cm. En ambas configuraciones (lente convergente y estenopeica), se requiere que el objeto se ubique a 20 cm del sistema óptico para proyectar una imagen de 2.5 cm en un plano a 5 cm de distancia. Sin embargo, la cámara con lente ofrece ventajas significativas:

- La imagen es más definida, enfocada y brillante.
- La lente permite mayor flexibilidad en el diseño óptico.
- La eficiencia de captación de luz es mucho mayor que en la cámara estenopeica.

2.3. Cámara Digital

Las cámaras fotográficas, como ya se mencionó en 1.1.1, han cambiado la forma en que se ve el mundo. Estas tienen el poder de guardar instantes que transcinden desde la perspectiva del fotógrafo y el observador. Si bien la fotografía análoga ha jugado, de igual forma un papel no menos relevante, las imágenes digitales popularizaron e hicieron omnipresente los ojos de las personas en muchos lugares. Las digitales no requieren proceso de revelado y están disponibles para su transmisión y almacenamiento de forma inmediata. Adicionalmente, el mercado las ha hecho accesibles a todos los presupuestos.

Fundamentalmente, la cámara digital es una versión extendida de la análoga que, en ambos casos, emulan el ojo humano como se ve al comparar la Figura 2.5 y Figura 2.2.

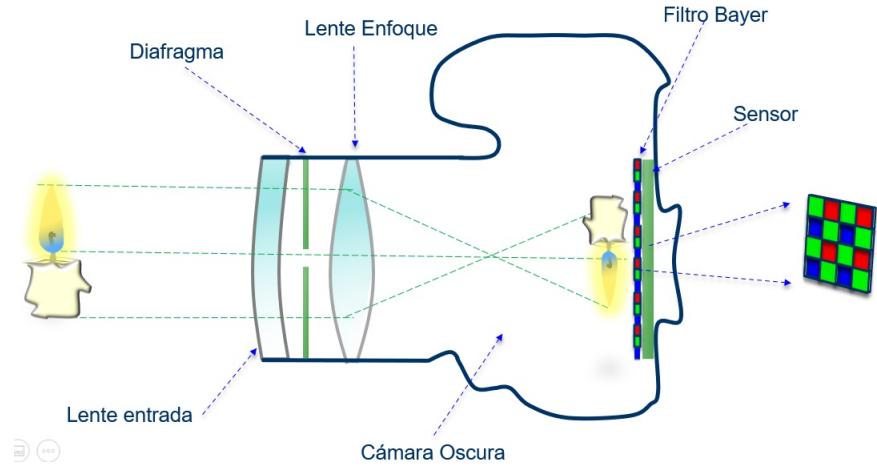


Figura 2.5: La cámara digital y su partes básicas.

La primera cámara digital reconocida fue diseñada por el ingeniero de Eastman Kodak Steven Sasson en 1975. Para lograrlo usó algunas partes de equipos Motorola, una lente de cámara de cine Kodak y algunos sensores electrónicos Fairchild CCD recién inventados. Su primer prototipo, como ya se indicó, era del tamaño de una tostadora grande, pesaba casi 4 kg y solo tomaba imágenes en escala de grises. Pese a las mejoras en la tecnología de sensores, su principio de funcionamiento se conserva aún. La cámara digital básica está formada por una caja oscura y algunas lentes que forman una imagen invertida sobre el plano de formación [41]. Allí se localiza un arreglo de sensores fotosensibles localizados de forma espacialmente discreta que reciben muestras de la intensidad de la luz que proyecta el objeto fotografiado. Las cámaras monocromáticas son aún ampliamente usadas en laboratorios porque permiten mejor resolución, son más sensibles a la luz y generan menor ruido [36]. Véase la Figura 2.6.

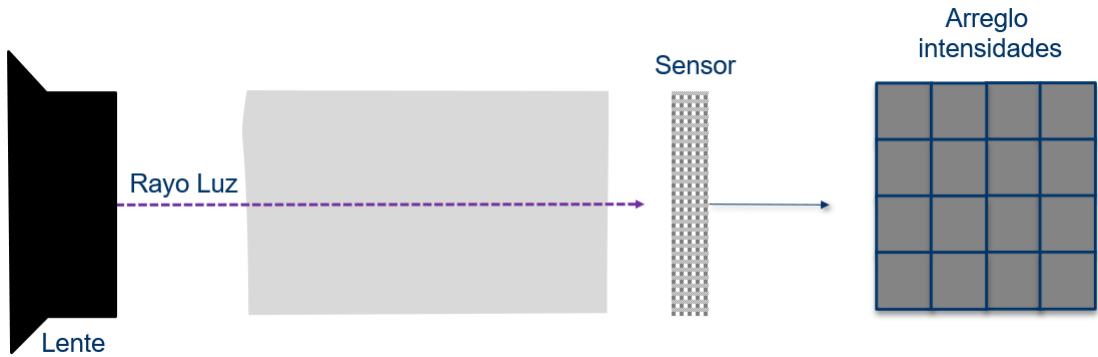


Figura 2.6: Principio de funcionamiento de una cámara digital.

2.4. Principio básico del color

Las cámaras digitales, por defecto, no tienen percepción del color. Para registrar diferentes longitudes de onda, se recurre a varios arreglos. Prácticamente todos los sensores digitales funcionan capturando la luz en un arreglo, de forma similar a una red de cubos que almacenaría la lluvia de colores que cae como se aprecia en la Figura 2.7. Cuando comienza la exposición, cada sensor se descubre con el obturador para recoger la luz entrante. Cuando finaliza la exposición, se lee como una señal eléctrica, que luego se cuantifica y almacena como un valor numérico en un archivo matricial que forma la imagen [12].

La imagen a color se logra usando una cámara de un solo arreglo monocromático de sensores como el usado en las imágenes a escala de grises. Para capturar la imagen a color, en tres diferentes momentos, se debe tomar una fotografía usando un filtro de color rojo, verde y azul respectivamente como se ve en la Figura 2.8. Esta aproximación a la

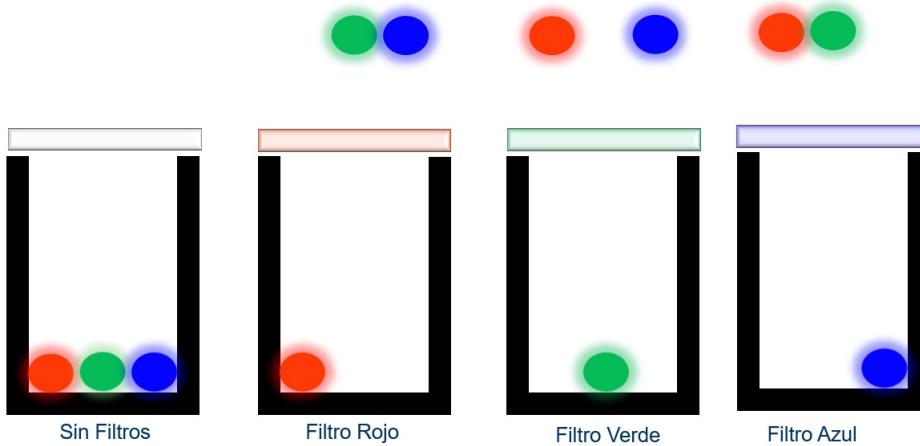


Figura 2.7: Representación de las celdas fotosensibles.

solución, si bien es elemental, ha sido usada altamente en aplicaciones donde el tiempo de exposición es relativamente largo entre cambios de filtro para cada toma.

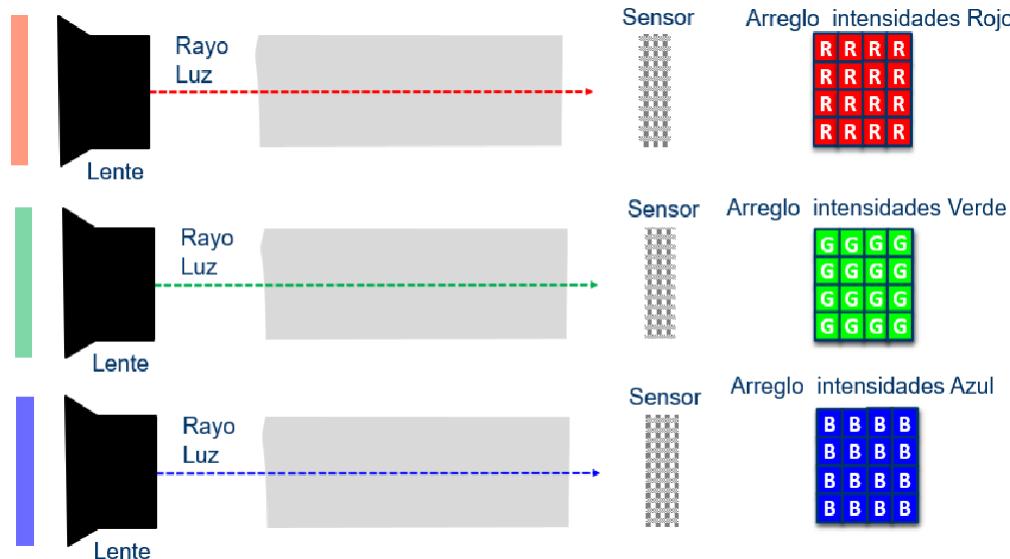


Figura 2.8: Artificio para tomar fotografías a color con una cámara monocromática. Para esto se usan filtros de colores y se guarda la secuencia de las tres fotografías roja, verde y azul.

2.4.1. Cámara con tres arreglos

La luz que forma la imagen pasa a través de un conjunto de prismas que dividen y reflejan la luz en 3 trayectorias, las cuales se filtran con películas translúcidas de color rojo, verde y azul en cada caso. Los sensores están alineados de tal forma que un rayo de luz que pasa a través de un prisma golpea el mismo píxel correspondiente en cada uno de los 3 sensores. La disposición ilustrada en la Figura 2.9 tiene como ventaja que los colores son más vivos, más naturales, y con mejores detalles en los bordes ya que no se requiere interpolación alguna. Como desventajas se destaca su mayor precio, su más alto volumen y lo vulnerable a desalineación de los sensores. No obstante, la mayoría de las cámaras de profesionales digitales de cine y televisión utilizan el sistema 3CCD.

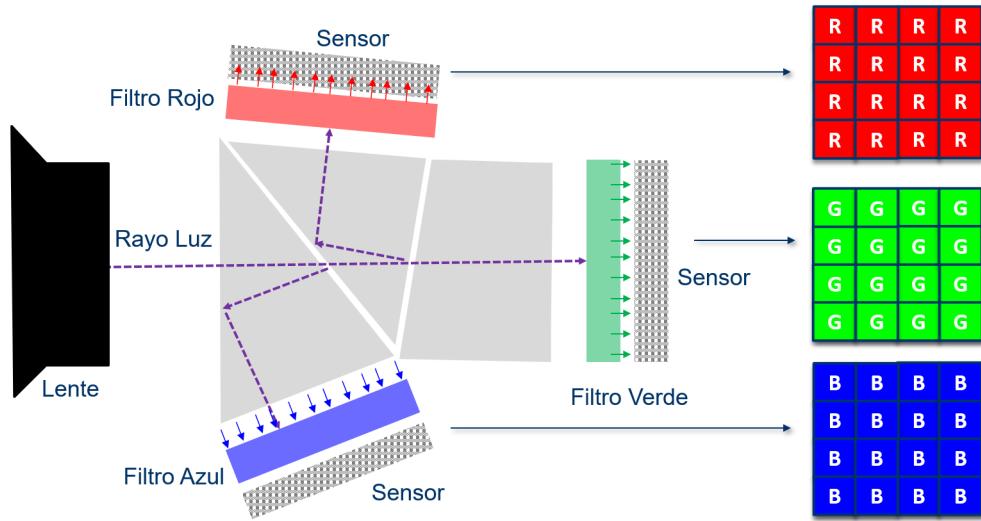


Figura 2.9: Arreglo para tomar fotografías a color con tres arreglos de sensores usando prismas translúcidos.

2.4.2. Cámara con un solo arreglo

Sus arreglos matriciales fotosensibles no discriminan entre longitudes de onda. Para lograr capturar los colores, existe una técnica simple pero efectiva basada en el filtro de Bayer.

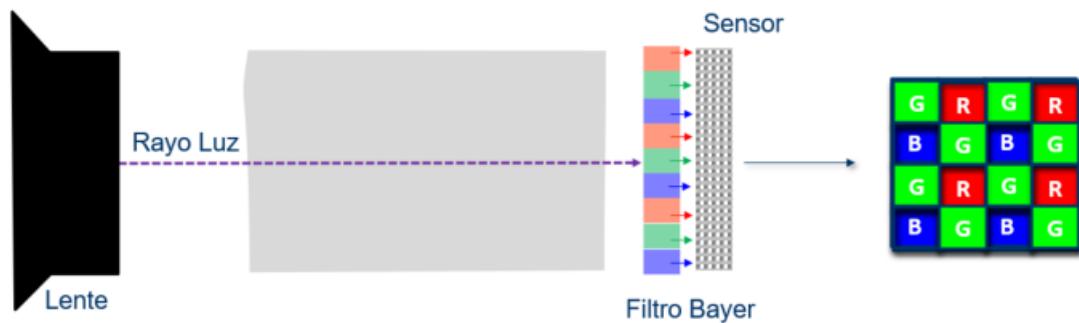


Figura 2.10: Principio de funcionamiento de una cámara digital a color con filtro Bayer.

Este es un arreglo de películas translúcidas de colores rojo, verde y azul que se colocan frente a un único arreglo de sensores monocromáticos en una distribución específica como se muestra en la Figura 2.10. Básicamente, este proceso replica los conos existentes en el ojo humano sobre la retina, por lo que hay 50 % filtros de verde, 25 % de azul y 25 % rojo. Véase la Figura 2.11.

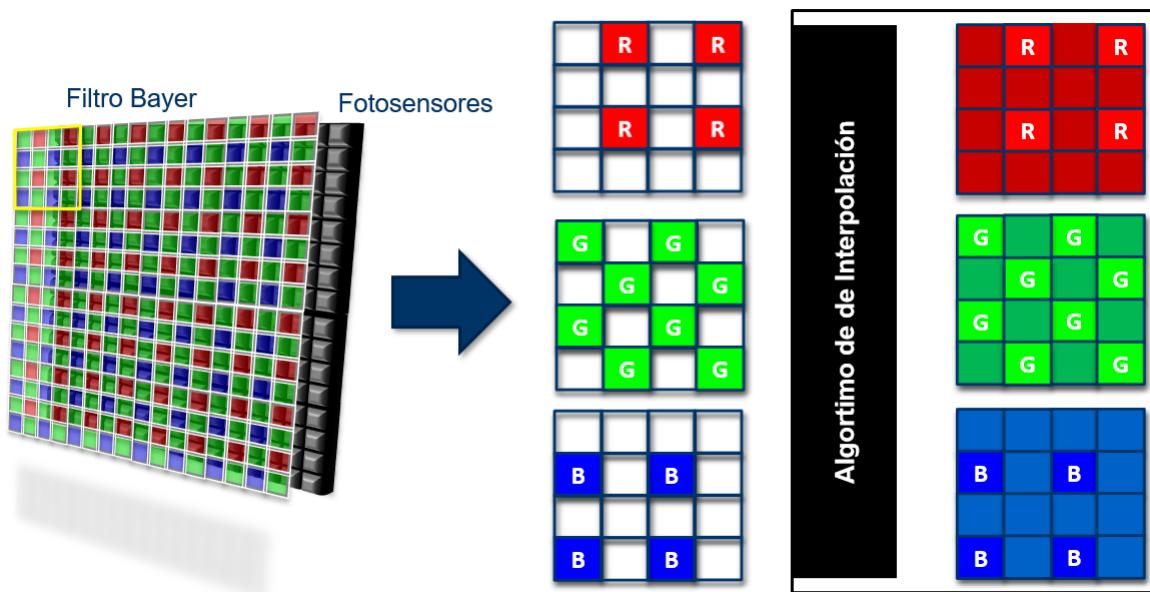


Figura 2.11: Cámara digital a color con filtro Bayer antes del algoritmo de interpolación y después de este.

En el proceso de formación de la imagen en la cámara, esta se proyecta sobre el fondo de forma invertida como se ve en la Figura 2.12. Cada sensor captura un poco de la luz entrante en las posiciones discretas, tomando solo uno de los tres colores y dejando faltante la información para la misma posición. Esta se conoce como la imagen RAW o cruda, la cual no tiene ninguna compresión ni procesamiento algorítmico. Para formar la fotografía completamente a color, se requiere interpolar la información de las posiciones faltantes sobre cada capa del color.

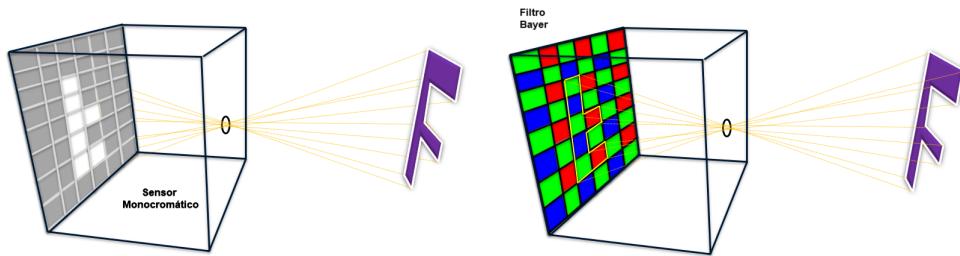


Figura 2.12: Principio de cámara oscura para formación de imágenes en escala de grises y a color con filtro Bayer.

Como ventaja principal del uso de arreglo con filtro Bayer se encuentra su bajo precio, al igual que el poco volumen que ocupa la disposición propuesta. Como desventaja se puede decir que está la baja exactitud del color y la pérdida de detalles sobre toda la imagen y especialmente en los bordes. Para efectos de ilustración, en la Figura 2.13 se presenta la imagen de Cartagena descompuesta en cada una de sus capas del color en formato Raw del filtro Bayer, Figura 2.13 (primera fila) y su versión interpolada y a color de estas, Figura 2.13 (segunda fila).

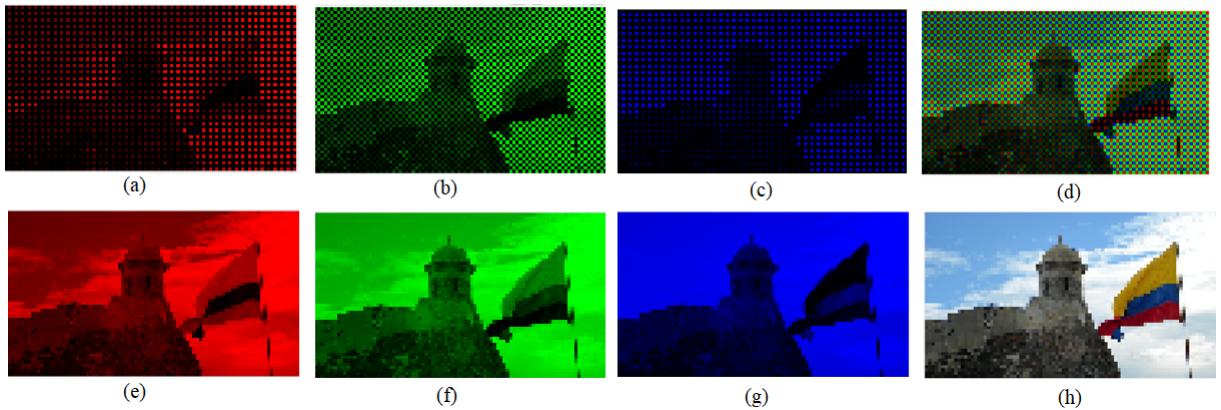


Figura 2.13: imagen de Cartagena descompuesta en cada una de sus capas del color en formato Raw del filtro Bayer (primera fila) y su versión interpolada y a color de estas (segunda fila).

La Figura 2.14 muestra visualmente la diferencia entre una imagen capturada con una cámara con filtro Bayer de un sensor y una de 3 sensores con arreglo de prismas [19].



Figura 2.14: Comparación entre una imagen tomada con una cámara de color tipo Bayer de 5 megapíxeles (1CCD, izquierda) y una cámara JAI AT-200 de 2 megapíxeles con sensor 3CCD (derecha). A pesar de que la cámara Bayer tiene mayor resolución, presenta contaminación de color, menor diferenciación entre colores similares y menor nitidez, debido al uso de filtros de polímero y el proceso de interpolación.

2.4.3. Ejemplo en Python: Separación de capas de color

Desarrolle un script en Python que cargue una imagen llamada 'cartagena.jpg' a color en formato RGB y extraiga sus componentes de color rojo, verde y azul. A continuación, este debe mostrar cada una de estas capas por separado, tanto en escala de grises como en su respectivo color original, utilizando subplots para organizar las visualizaciones en una única ventana.

Solución

```

1 import matplotlib.pyplot as plt    # Importar librería para gráficos
2 import numpy as np                # Importar librería para manejo numérico
3
4 RGB = plt.imread("cartagena.jpg")  # Leer imagen RGB

```

```

5
6 r = RGB[:, :, 0] # Canal rojo
7 g = RGB[:, :, 1] # Canal verde
8 b = RGB[:, :, 2] # Canal azul
9
10 # Mostrar imagen original y canales en escala de grises
11 plt.figure(figsize=(10, 5))
12 plt.subplot(2, 2, 1); plt.imshow(RGB); plt.title("Color"); plt.axis('off')
13 plt.subplot(2, 2, 2); plt.imshow(r, cmap='gray'); plt.title("Rojo"); plt.axis('off')
14 plt.subplot(2, 2, 3); plt.imshow(g, cmap='gray'); plt.title("Verde"); plt.axis('off')
15 plt.subplot(2, 2, 4); plt.imshow(b, cmap='gray'); plt.title("Azul"); plt.axis('off')
16
17 # Crear imágenes con un solo canal visible (color)
18 R = np.copy(RGB); R[:, :, 1] = 0; R[:, :, 2] = 0 # Solo rojo
19 G = np.copy(RGB); G[:, :, 0] = 0; G[:, :, 2] = 0 # Solo verde
20 B = np.copy(RGB); B[:, :, 0] = 0; B[:, :, 1] = 0 # Solo azul
21
22 # Mostrar imagen original y canales en color
23 plt.figure(figsize=(10, 5))
24 plt.subplot(2, 2, 1); plt.imshow(RGB); plt.title("Color"); plt.axis('off')
25 plt.subplot(2, 2, 2); plt.imshow(R); plt.title("Rojo"); plt.axis('off')
26 plt.subplot(2, 2, 3); plt.imshow(G); plt.title("Verde"); plt.axis('off')
27 plt.subplot(2, 2, 4); plt.imshow(B); plt.title("Azul"); plt.axis('off')
28
29 plt.show() # Mostrar todo

```

En la Fig.2.15 se muestra el resultado de la ejecución del código en Python con las imágenes en escala de grises (a la izquierda) y a Color (a la derecha) de la imagen de 'cartagena.jpg'.

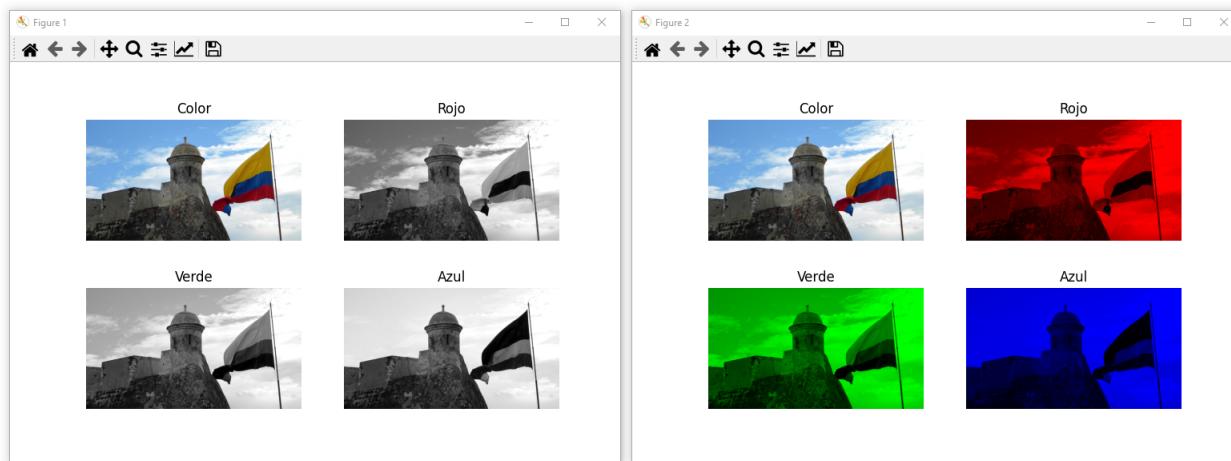


Figura 2.15: Resultados de la ejecución del código del ejemplo con las imágenes en escala de grises (a la izquierda) y a color (a la derecha) del ejemplo en Python.

2.5. Funciones

Estas son algunas funciones que resultan útiles en este capítulo. La conversión entre imágenes en formato Bayer (raw) y RGB es fundamental en el procesamiento de imágenes digitales adquiridas por sensores con filtros de color. El patrón de mosaico Bayer define cómo se distribuyen los colores rojo, verde y azul sobre la matriz del sensor. A continuación, se presentan las funciones más comunes asociadas a este proceso:

1. `demosaic(I, sensorAlignment)`: Convierte una imagen **Bayer** monocromática (una sola banda) en una imagen RGB a color. La función utiliza un algoritmo de interpolación para estimar los valores faltantes de cada canal (R, G, B) en función del patrón definido por `sensorAlignment`. Este patrón puede ser uno de los siguientes: 'RGGB', 'BGGR', 'GRBG' o 'GBRG', los cuales indican la disposición de colores en la matriz de píxeles del sensor. El resultado es una imagen a color reconstruida a partir del patrón de filtro Bayer.

2. `rgb2bayer(I, sensorAlignment)`: Simula la conversión inversa de una imagen RGB a su equivalente en formato Bayer. Esta operación es útil para pruebas de algoritmos de demosaicing y análisis de sensores de imagen.

Esquemas de Alineación del Sensor Bayer

La variable `sensorAlignment` especifica la disposición de los filtros de color en el sensor de la cámara. Los patrones más comunes son:

- 'RGGB': Rojo en la esquina superior izquierda, seguido por verde, verde y azul.
- 'BGGR': Azul en la esquina superior izquierda, seguido por verde, verde y rojo.
- 'GRBG': Verde-rojo en la primera fila, verde-azul en la segunda.
- 'GBRG': Verde-azul en la primera fila, verde-rojo en la segunda.

Estos patrones afectan directamente el proceso de reconstrucción del color (demosaicing) y deben ser especificados correctamente para asegurar la fidelidad cromática de la imagen resultante.

2.6. Preguntas

2.6.1. Preguntas sobre hechos del texto:

1. ¿Qué parte del espectro electromagnético corresponde a la luz visible?
2. ¿Qué células sensibles a la luz se encuentran en la retina del ojo humano?
3. ¿Quién construyó la primera cámara digital de la historia?
4. ¿En qué consiste el filtro de Bayer usado en cámaras digitales?
5. ¿Qué es una imagen RAW en el contexto de las cámaras digitales?

2.6.2. Preguntas de análisis y comprensión sobre el texto:

1. ¿Cómo se explica la naturaleza dual de la luz según las teorías físicas?
2. ¿Por qué el ojo humano percibe diferentes colores según la longitud de onda de la luz?
3. ¿Cuáles son las ventajas y desventajas de los sistemas de cámara con 1 sensor frente a los de 3 sensores?
4. Analiza las semejanzas y diferencias entre el funcionamiento del ojo humano y una cámara digital.
5. Explica la técnica de usar filtros de color en una cámara monocromática para obtener imágenes a color.

Capítulo 3

La Imagen

3.1. La imagen

Una imagen es una representación esculpida, pintada o fotografiada de un objeto o escena (Véase la Figura 3.1). En esencia, una imagen busca generar una impresión visual análoga a la realidad, ya sea de forma realista o simbólica, por lo que resulta ser una herramienta fundamental para la comunicación, el arte, el diseño y el acceso al conocimiento, dado que puede representar visualmente conceptos, ideas, datos y elementos del mundo real o imaginario [13].



Figura 3.1: Ejemplos de imágenes. a) Escultura localizada en el Museo Metropolitano de Arte de Nueva York. b) Pintura encontrada en en Museo del Vaticano. c) Fotografía tomada a la entrada del Museo del Louvre en París.
Fuente: Los autores.

3.2. La Imagen Digital

Se refiere a la representación que se ha creado a través de tecnología electrónica, tal como un escáner o una cámara digital. Esta no se puede lograr sin el software y el hardware adecuado que permite hacerle visible. Más formalmente, una imagen digital es una representación en *2D de una escena en 3D* que ha sido discretizada tanto en coordenadas espaciales como en la profundidad del brillo del color para cada punto. Matemáticamente se representa como una matriz en dos dimensiones $I(x, y)$ de números, o una serie de matrices, cuando se requiere una para cada banda del color. El valor del brillo se denomina nivel de gris y a cada elemento de la matriz se le llama *pixel* o *pel*, palabra derivada del término "*picture element*" [13].

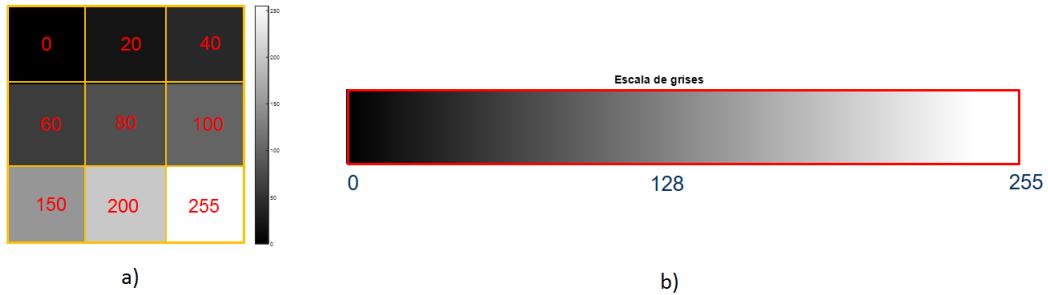


Figura 3.2: a) Muestra de píxeles representados en escala de grises. b) Escala de grises de 0 a 255.

Cada píxel representa el color (o nivel de gris) en un único lugar de la imagen, por lo que un píxel es similar a un pequeño punto de un único color particular. Los píxeles son similares al grano encontrado en una imagen fotográfica convencional impresa, pero dispuestas en un patrón regular de filas y columnas (Véase la Figura 3.3). Una imagen digital en escala de grises se puede representar como se muestra en la ecuación 3.1:

$$I(x, y) = \begin{bmatrix} I(1, 1) & I(1, 2) & \cdots & I(1, M) \\ I(2, 1) & I(2, 2) & \cdots & I(2, M) \\ \vdots & \vdots & \ddots & \vdots \\ I(N, 1) & I(N, 2) & \cdots & I(N, M) \end{bmatrix} \quad (3.1)$$

Con $0 \leq I(x, y) \leq k - 1$, donde $k = 2^n$ es siempre potencia de 2 y n es el número de bits. De igual manera, se acostumbra a representar a N (*Número de filas*) y M (*Número de columnas*), pero no les es obligatorio. Algunos valores típicos son $n = 8$, $k = 256$, $N = 480$ y $M = 640$.

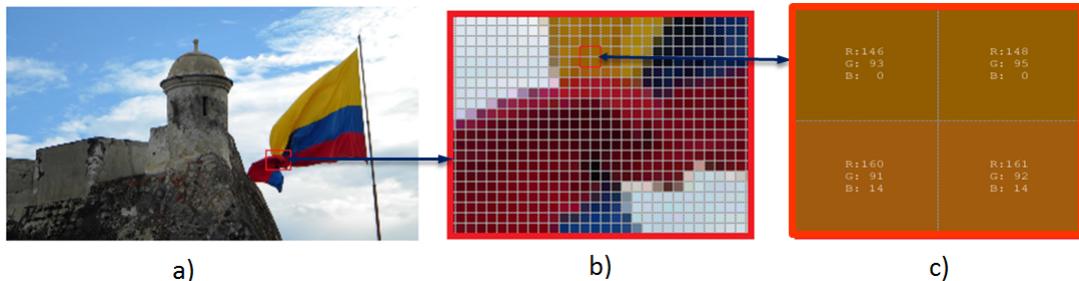


Figura 3.3: Ejemplos de imagen digital. a) imagen digital con alta detalle. b) imagen digital con aproximación donde se visualizan cada uno de los píxeles. c) Algunos píxeles de la aproximación con su valor de las componentes R (rojo), G (verde) y B (azul).

3.3. La Imagen a color

La mayoría de las imágenes a color están con base en los colores primarios rojo, verde y azul (RGB), normalmente haciendo uso de $n = 8$ bits para cada componente del color. En las imágenes a color del modelo RGB, cada pixel requiere $3 \times 8 = 24$ bits para codificar todas las tres componentes, con un rango de brillo de cada pixel entre [0 255]. En la Figura 3.2 se muestra la escala de grises usualmente usada para representar las posibles tonalidades de un color. Ella está dividida en 256 partes para una representación de 8 bits [13].

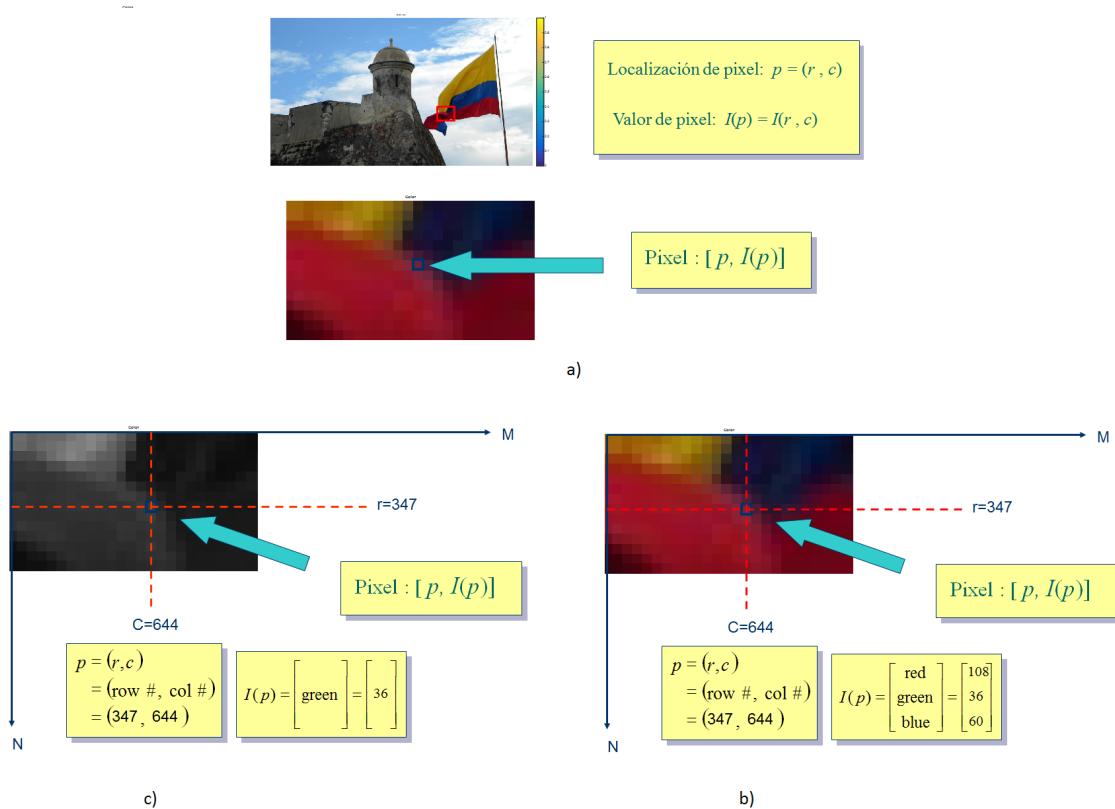


Figura 3.4: Descomposición por pixeles de una imagen a color. a) Imagen a color. b) Pixel de una imagen a color con tres intensidades. c) Pixel de una capa de la imagen a color con una sola intensidad.

Si bien la mayoría de imágenes a color contienen tres matrices, una para cada componente roja, verde y azul en RGB, estas pueden ser de cuatro o más capas de información. Algunos ejemplos de aplicación son la preimpresión con base en el modelo de color CMYK (Cian, Magenta, Amarillo y Negro) y las fotografías multiespectrales que contienen información en una de sus capas adicionales, más allá de lo visible al ojo humano.

3.3.1. Ejemplo en Python: Creación de pixeles de colores

Se propone un script genera y visualiza una imagen RGB de 3x3 píxeles con los siguientes colores: rojo, verde, azul, cian, amarillo, magenta, negro, blanco y gris. Utilizando matrices NumPy, se definen los valores de los canales rojo, verde y azul para cada píxel, representando los colores deseados. Finalmente, la imagen se visualiza con Matplotlib

Solución

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 #tamaño de las matrices a visualizar
5 size=(3,3,3)
6 #Una matriz de ceros 3x3
7 RGB=np.zeros(size)
8 RGB=np.uint8(RGB)
9
10 #Asignar Valores a cada pixel
11 RGB[:, :, 0]=([[0,255,255],[255,128,0],[255,0,0]])
12 RGB[:, :, 1]=([[255,255,0],[0,128,255],[255,0,0]])
13 RGB[:, :, 2]=([[255,255,0],[255,128,0],[0,0,255]])
14 plt.figure(figsize=(5,5))
15 plt.imshow(RGB)
16 plt.show()

```

En Fig.3.5 se muestra el resultado de la ejecución del script en Python.

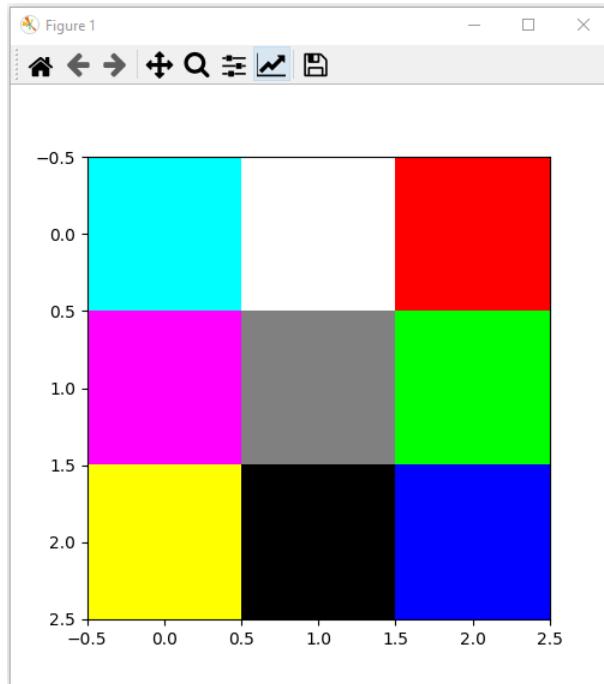


Figura 3.5: Resultado de la ejecución de código de Python para generar colores en una imagen a partir del modelo RGB. En ella se aprecian los colores rojo, verde, azul, cian, amarillo, magenta, negro, blanco y gris.

3.4. La Resolución Espacial

En rigor esta se define como la densidad de píxeles presentes en una imagen por unidad de longitud. Cuanto mayor sea la resolución, más información se posee de la imagen y mejor es su representación. Si se mantiene el mismo tamaño de la imagen y se aumenta la cantidad de píxeles para la misma, la imagen se vuelve más definida. Es de destacar que la resolución se mide usualmente en píxeles por pulgada (dpp). Algunas veces, los usuarios solo están interesados en el número total de píxeles de la imagen, por lo que es habitual usar el término resolución para hacer referencia al número de píxeles por filas y columnas de una imagen. En Fig-3.6 se ilustra la imagen del Castillo de San Felipe en Cartagena, Colombia, con diferentes resoluciones espaciales [13].

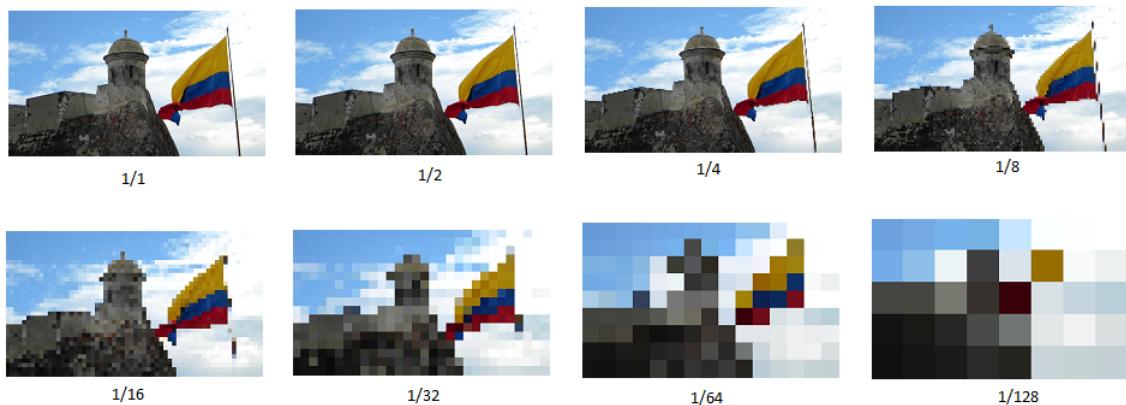


Figura 3.6: Imagen a diferentes resoluciones espaciales

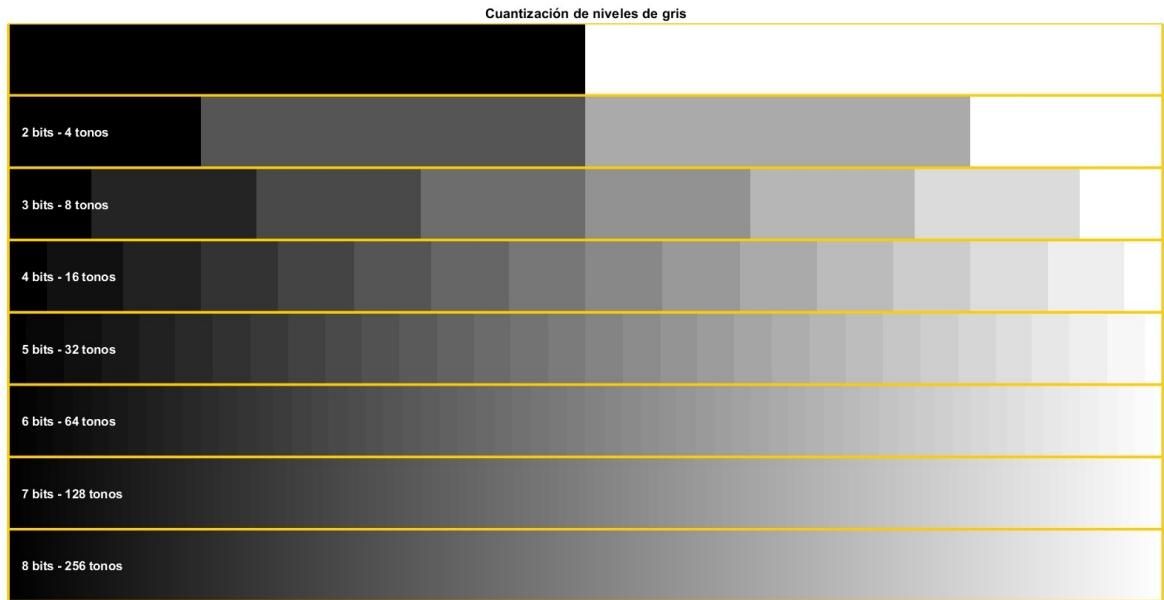


Figura 3.7: Escala de tonalidades con diferentes números de bits.

3.5. La profundidad del Color

La profundidad del color cuantifica la cantidad de tonos únicos que la imagen puede presentar adecuadamente. Esto no significa que la imagen utilice necesariamente todos estos colores, sino que puede mostrar colores con ese nivel de detalle. Usualmente, la estimación de la cantidad de tonalidades que puede mostrar se realiza en función de la base binaria $k = 2^n$ como se ilustra en la Figura 3.7 [13].

Un ojo humano sano tiene tres tipos de células del color, denominadas conos, cada una de las cuales puede discriminar alrededor de 100 tonalidades diferentes. A partir de ello, los investigadores estiman que la cantidad de colores que se pueden distinguir en alrededor de un millón (100^3). No obstante, la percepción del color es una habilidad altamente subjetiva que varía entre las personas y puede variar ligeramente de una persona a otra debido a factores genéticos y fisiológicos. Por lo anterior, guardar una imagen en más de 24 bits es excesivo si el único propósito previsto es la visualización; las imágenes con 24 bits son más que suficiente para un eficiente procesamiento. En la Figura 3.8, se pueden apreciar el efecto de la cantidad de bits en términos de la presentación de la tonalidad.



Figura 3.8: Imagen a diferentes profundidades de bits del color.

En la Tabla 3.1 se ilustra la cantidad de tonalidades que se pueden representar dado el número de bits de la imagen. Como se puede evidenciar, 8 bits generan 256 tonalidades. Para el caso de una imagen en el modelo de color RGB, cada capa del color usara 8 bit, por lo que en total lleva a una representación de 24 bits. Esto conduce a tener

Profundidad (bits)	Tonalidades (cantidad)
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256

Cuadro 3.1: Cantidad de tonalidades que se pueden representar dado el número de bits de una imagen.

16.777.216 colores (2^{24}).

3.6. Algunos Formatos Gráficos

Se presentan algunos de los formatos gráficos más utilizados en el procesamiento de las imágenes [32].

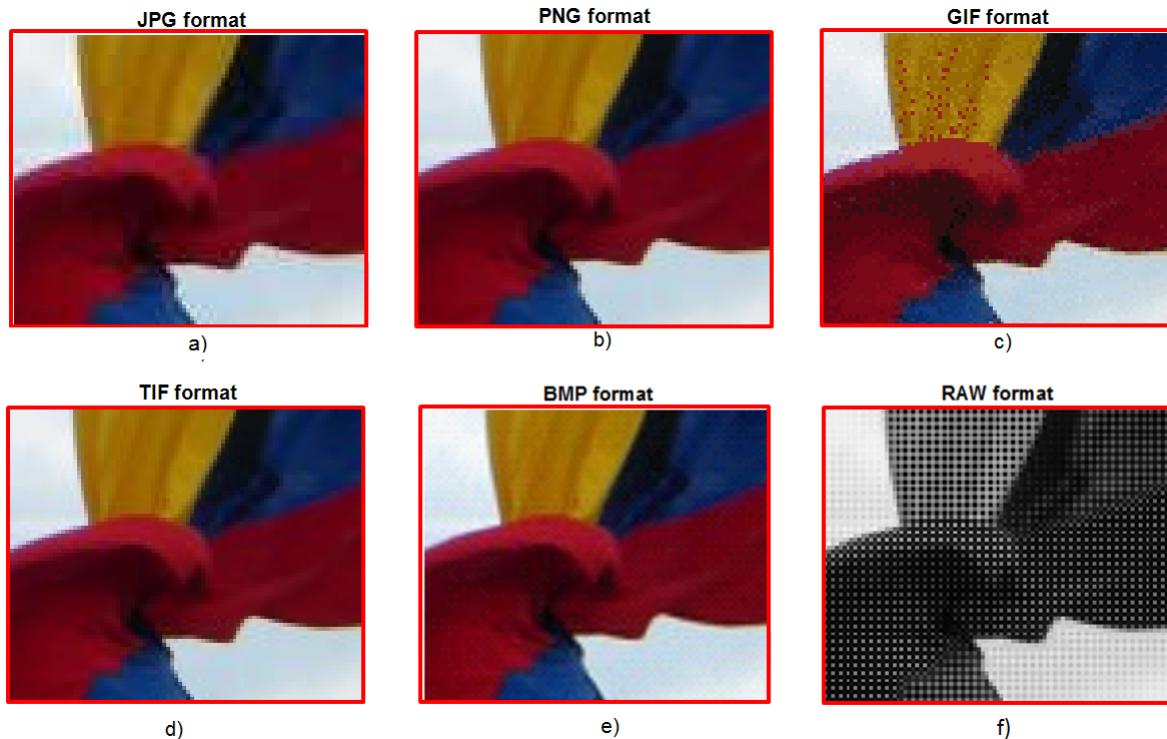


Figura 3.9: Detalle que evidencia los efectos de cada formato sobre una imagen. a) Imagen en formato JPG con artefactos. b) Imagen en formato PNG. c) Imagen en formato GIF con reducción del color. d) Imagen en formato TIF. e) Imagen en formato BMP sin compresión. f) Imagen en formato RAW sin reconstrucción.

3.6.1. JPEG

Esta es la abreviatura de Joint Photographic Experts Group. Los archivos JPEG pierde mucha información de la imagen original debido a sus altos niveles de compresión, lo cual se debe a que JPEG descarta la mayor parte de la

información para mantener el tamaño del archivo de imagen pequeño sacrificando su calidad, lo que puede causar la aparición de artefactos. Este es el formato más popular de los utilizados en la web (Véase la Figura 3.9 parte a).

Pros de JPEG

- Color de 24 bits, con más de 16 millones, lo que le hace ideal para fotografías que requieran un gran detalle.
- Es el formato de imagen más utilizado y más ampliamente aceptado.
- Compatible en la mayoría de los sistemas operativos (Mac, PC, Linux).

Contras de JPEG

- Descarta una gran cantidad de datos después de la compresión.
- Tiende a crear artefactos.
- No se pueden hacer animaciones.
- No soporta las transparencias.

3.6.2. PNG

Portable Network Graphics es un formato introducido como un estándar desde 1996. Es un formato de imagen específicamente diseñado para la web, ya que cubre algunos de los mejores aspectos manejados por GIF, ya que puede guardar con 256 colores como máximo, aunque puede soportar hasta 24 bits, pero de manera más eficiente. De igual forma, soporta una transparencia de 8 bits. PNG fue creado con la intención de reemplazar el formato GIF ya que no requiere una licencia de patente para su compresión (Véase la Figura 3.9 parte b).

Pros de PNG

- De bajas pérdidas, a pesar de su compresión.
- Soporta la transparencia mejor que GIF.

Contras de PNG

- Tienden a generar archivos más grandes que el JPEG.
- No soporta animaciones.
- No es soportado por todos los navegadores web.

3.6.3. GIF

Es la abreviatura de Graphics Interchange Format. Este formato está limitado a una paleta de 8 bits con sólo 256 colores. GIF sigue siendo un formato de imagen muy popular en el Internet gracias a que su tamaño de la imagen es relativamente pequeño respecto a otros. GIF comprime las imágenes de dos maneras: primero, reduciendo el número de colores y, de esta forma, el número de bits por píxel. En segundo lugar, GIF sustituye múltiples patrones de color que ocurrán, así que en lugar de almacenar varias intensidades de un color, guarda solo una. GIF es más adecuado para gráficos, diagramas, caricaturas y logotipos con relativamente pocos colores. Este sigue siendo el formato más elegido para efectos de animación en la Internet. Una muy importante característica es su entrelazado, el cual da la ilusión de que gráfico carga más rápidamente; de esta forma, cuando la imagen se carga en un navegador, el GIF aparece primero borroso, pero a medida que se descargan más datos, la imagen se vuelve más definida (Véase la Figura 3.9c).

Pros de GIF

- Soporta transparencias.
- Permite hacer pequeños efectos de animación.
- Calidad "sin pérdidas": contienen la misma cantidad de calidad que el original, excepto que ahora sólo tiene 256 colores ideal para imágenes con colores limitados, o con regiones planas de color.

Contras del GIF

- Solo admite 256 colores.
- Es el formato más antiguo de la web, habiendo existido desde 1989.
- No ha sido actualizado ya que, a veces, el tamaño del archivo es mayor que PNG.
- Tiene propiedad en su algoritmo de compresión.

3.6.4. TIFF

La especificación TIFF (Tag Image File Format) fue originalmente lanzada en 1986 por Aldus Corporation como un método estándar para almacenar imágenes en blanco y negro creadas por escáneres y aplicaciones de edición electrónica.. Fue posteriormente en 2009 transferido al dominio de Adobe Systems quien posee todos sus derechos. TIFF es popular entre los usuarios comunes, pero ha ganado reconocimiento en el diseño gráfico, la publicación y la industria de la fotografía por lo que TIFF es un formato rico y soportado por muchos programas de imágenes. Es muy flexible y puede ser de baja pérdida (Véase la Figura 3.9 parte d).

Pros de TIFF

- Es muy flexible ya que soporta varios tipos de compresión como JPEG, LZW, ZIP o ninguna compresión en absoluto.
- Es un formato de alta calidad ya que almacena todo el color y la información.
- El formato TIFF puede guardarse por capas.

Contras de TIFF

- Tamaño de archivo muy grande.
- Gran consumo de espacio en disco.
- Requiere mucho tiempo para su descargas en la web.

3.6.5. BMP

Los archivos de mapas de bits BMP (bitmap) de Windows son archivos de imagen que forman parte del sistema operativo Microsoft Windows, siendo uno de los primeros formatos de imagen. El formato de mapa de bits original creado para Windows 1.0 era muy simple. Tenía una paleta de colores fija sin compresión de datos. Este primer formato fue diseñado para soportar las tarjetas gráficas más populares de los PC IBM en uso en aquel momento (CGA, EGA, Hercules y otros). El formato original era conocido como mapa de bits dependiente del dispositivo original de Windows (device-dependent bitmap DDB). Los archivos BMP son usualmente grandes y no tienen compresión, pero se favorecen las imágenes ya que estas son ricas en color, de alta calidad, simples y compatibles en varios programas de Windows y sistemas operativos no necesariamente Microsoft. Los archivos BMP no son actualmente muy populares, ya que no son compatibles en todas las plataformas operativas ni funcionan bien en la web (Véase la Figura 3.9 parte e) .

Pros de BMP

- Funciona bien con la mayoría de los programas de Windows y su sistema operativo.
- No tiene artefactos.
- Imágenes ricas en color.

Contras de BMP

- No escala ni comprime.
- Los archivos son muy grandes, lo que no es amigable con la web.

3.6.6. RAW

El formato gráfico RAW contiene información de píxeles procedente directamente desde el arreglo de sensores de la cámara digital. RAW requiere someterse a una reconstrucción de algunos píxeles ya que no contiene información de los tres colores RGB para cada pixel. Las cámaras digitales normalmente revelan el archivo RAW convirtiéndolo en un archivo de imagen más popular como el JPEG o el TIFF a todo color para luego almacenar el archivo en su tarjeta de memoria. El formato de archivo RAW puede utilizar una compresión sin pérdidas, por lo que no sufre de los artefactos de compresión visibles (Véase la Figura 3.9 parte f y El Cuadro 3.2) .

Imagen de 539 × 958	Formato	Espacio en disco
Cartagena.jpg	JPEG	32.2Kb
Cartagena.png	PNG	780Kb
Cartagena.gif	GIF	311Kb
Cartagena.tif	TIFF	614Mb
Cartagena.bmp	BMP	1.47Mb

Cuadro 3.2: Comparativo del almacenamiento requerido para la misma imagen del castillo de San Felipe en diferentes formatos.

3.7. Funciones

Las siguientes funciones están relacionadas con las operaciones básicas de lectura, visualización, escritura y separación de canales en imágenes digitales:

1. `imshow(I, ax=None, title=None)`: Muestra la imagen `I`. Si se proporciona el parámetro opcional `ax`, la imagen se dibuja en el eje indicado de Matplotlib. El título de la imagen puede incluirse con el parámetro `title`.
2. `imread(path)`: Lee una imagen desde un archivo gráfico ubicado en la ruta especificada por `path`. La imagen se devuelve como un arreglo `numpy` en formato RGB. Es compatible con múltiples formatos gráficos como PNG, JPEG, BMP, entre otros.
3. `imsplit(I)`: Separa los canales de color rojo, verde y azul de una imagen RGB `I`. Retorna tres matrices correspondientes a cada componente de color, lo cual es útil para procesar o visualizar individualmente cada canal.
4. `imwrite(path, I)`: Guarda la imagen `I` en la ruta especificada por `path`. El formato del archivo se determina por la extensión (.png, .jpg, .bmp, etc.) incluida en el nombre del archivo. Esta función permite exportar imágenes procesadas para su almacenamiento o distribución.

3.8. Preguntas

3.8.1. Preguntas de Comprensión

1. ¿Qué es una imagen digital?
2. ¿Cómo se representa matemáticamente una imagen digital?
3. Explica el concepto de resolución espacial.
4. Define la profundidad del color.
5. Diferencia entre RAW, TIFF, JPEG, PNG y GIF (Hacer una tabla).

3.8.2. Preguntas de Aplicación y Evaluación

1. ¿Cuál es el proceso y los métodos más comunes para convertir una imagen a color a una imagen en escala de grises?
2. ¿Qué factores se deben considerar al elegir el formato de imagen para un sitio web, y cuáles son los más recomendados? Por qué?
3. ¿Cómo afecta el aumento de la resolución espacial la calidad de una imagen? Solo el aumento de la resolución es suficiente?
4. ¿Cómo se determina la asignación adecuada de bits para representar una imagen en color y qué impacto tiene esto sobre su calidad general?
5. ¿Qué criterios deben tenerse en cuenta al seleccionar un formato de imagen para el análisis científico, y cuáles son los formatos más utilizados en este ámbito? Si es necesario, consultar.

Capítulo 4

Ajuste y Ecualización

4.1. El Histograma de una imagen

El histograma de una imagen es un vector que representa la frecuencia de la ocurrencia de cada uno de los valores de intensidad de los píxeles que hay en ella. El histograma h para una imagen I en escala de grises con valores de intensidad $I(u,v) \in [0, K - 1]$ contendría exactamente K entradas, siendo el caso para una imagen en escala de grises de 8 bits típica, $K = 2^8 = 256$. Cada entrada individual del histograma se define $h(i)$ como la cantidad de *píxeles* en la imagen I con valor de intensidad i , para todo $0 \leq i < K - 1$. De manera formal se puede escribir como se ilustra en la Ecuación (4.1) [5].

$$h(i) = \text{card} \{(x,y) \mid I(x,y) = i\}. \quad (4.1)$$

Donde:

- card denota la cardinalidad del conjunto, es decir, el número de elementos (píxeles, en este caso) que cumplen con la condición especificada.
- (x,y) son las coordenadas de los píxeles en la matriz de la imagen, donde x es la fila y y la columna.
- $I(x,y)$ es el valor de intensidad del píxel en la posición (x,y) .
- i es un valor de intensidad específico para el cual se está contando la cantidad de píxeles.

Se puede definir el histograma $h(i)$ como la suma de la delta de Kronecker para cada valor de intensidad i en la imagen:

$$h(i) = \sum_{x=1}^M \sum_{y=1}^N \delta(I(x,y) - i)$$

Donde M es el número total de filas y N es el número total de columnas de la matriz. La delta de Kronecker $\delta(z)$ se define como:

$$\delta(z) = \begin{cases} 1, & \text{si } z = 0 \\ 0, & \text{en otro caso} \end{cases}$$

Por lo tanto, en el contexto del histograma, la delta de Kronecker se utiliza para contar cuántas veces aparece un valor de intensidad en la imagen.

El histograma es una herramienta fundamental en el procesamiento de imágenes y análisis de visión artificial porque proporciona una representación gráfica de la distribución tonal de una imagen, lo que puede ser utilizado para operaciones como la mejora de contraste, umbralización automática, y para entender las características generales de la iluminación dentro de una imagen.

Por lo tanto $h(0)$ es el número de píxeles con el valor de intensidad 0, $h(1)$ el número de píxeles con el valor 1, y así sucesivamente. Finalmente, $h(255)$ es el número de todos los píxeles blancos que resultan ser los de intensidad máximo 255. El resultado del cálculo es un histograma unidimensional del vector h de longitud K . La Figura 4.1 muestra un ejemplo de una histograma de intensidad de la imagen I [5].

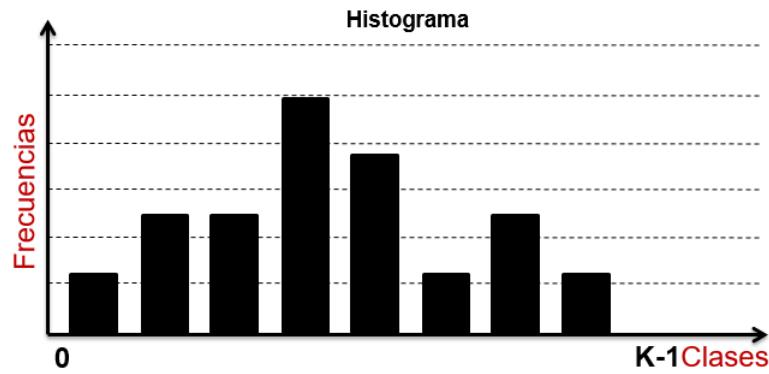


Figura 4.1: Ejemplo de histograma de frecuencias.

En el histograma de ocurrencias se debe cumplir que la suma todas las ocurrencias $h(i)$, para cada posible intensidad i , es igual al tamaño total de píxeles de la imagen I , donde M es el número de filas y N de columnas como se muestra en (4.2):

$$\sum_{i=0}^{K-1} h(i) = M \times N \quad (4.2)$$

De igual forma, también es frecuente dividir el histograma $h(i)$ por el tamaño total de la imagen ($N \times M$) para que este represente la probabilidad de ocurrencia $p(i)$ de cada intensidad i como se ilustra en (4.3). Con ello se logra que el área bajo el histograma sea igual a 1. Véase (4.4).

$$p(i) = \frac{h(i)}{M \times N} \quad (4.3)$$

$$\sum_{i=0}^{K-1} p(i) = 1 \quad (4.4)$$

Dado que un histograma no codifica información alguna de donde se originaron cada una de sus entradas individuales, los histogramas no contienen información sobre la disposición espacial de los píxeles de la imagen. Un histograma no permite reconstruir la imagen original que le generó, por lo que muchas imágenes pueden ser totalmente diferentes y poseer el mismo histograma. La Figura 4.2 muestra dos fotografías que poseen idénticos histogramas, pese a que las posiciones de los píxeles entre ambas no son iguales.

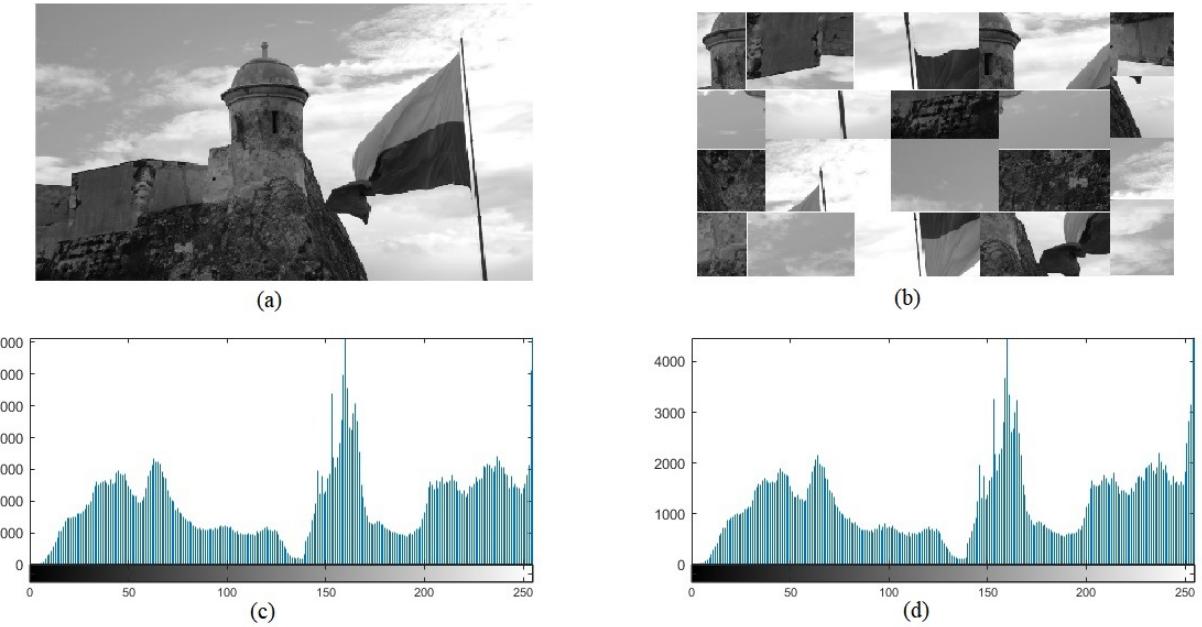


Figura 4.2: Histogramas de dos imágenes diferentes en escala de grises que poseen igual histograma.

El histograma toma más sentido cuando se realiza sobre cada una de las capas del modelo del color elegido, como es el caso de la Figura 4.3. El análisis del histograma toma suprema importancia en la interpretación estadística de la imagen ya que permite estimar criterios de mejora o restauración.

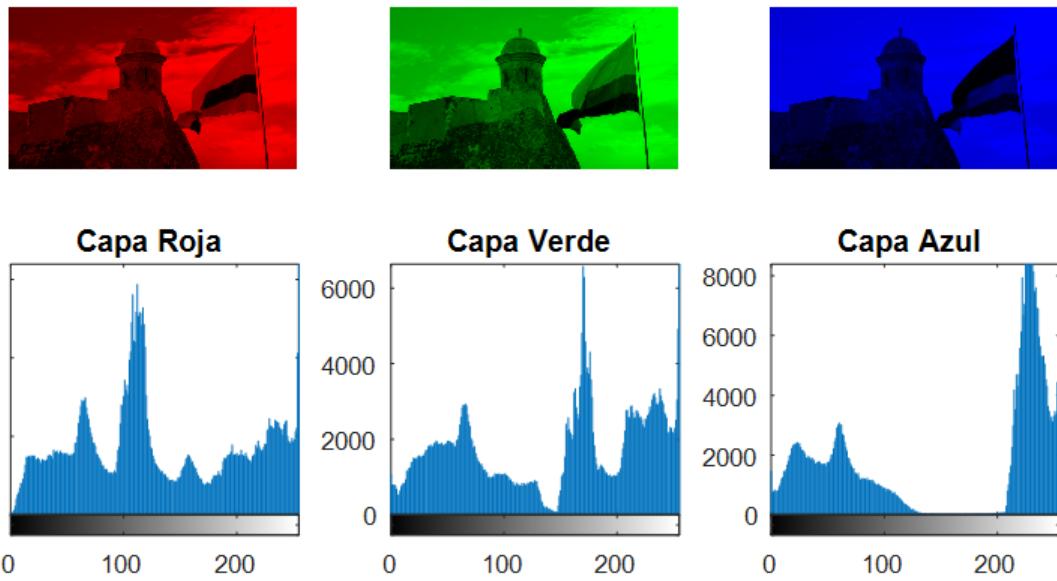


Figura 4.3: Imagen en escala de grises de 8 bits por color y su respectivo histograma para cada capa del modelo RGB.

El histograma tiene muchos usos en el procesamiento de imágenes. Con solo mirarlo se puede obtener mucha información sobre algunas características de la imagen tales como el brillo y el contraste. Como se verá más adelante,

él también resulta útil a la hora de escoger el umbral de segmentación de la imagen. En la Figura 4.4 se muestran algunos ejemplos del análisis del brillo y el contraste a partir del histograma.

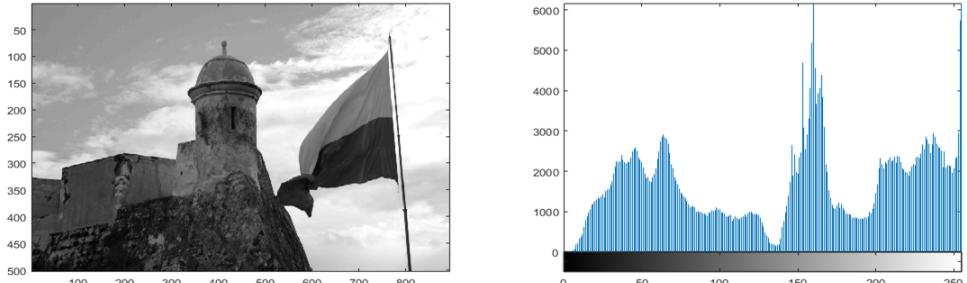
4.1.1. Ejemplo en Python: Extracción de capas e Histogramas

El código propuesto en Python ilustra cómo cargar la imagen de 'cartagen.jpg', reducir su tamaño para agilizar el procesamiento, descomponerla en sus canales de color rojo, verde y azul, y analizar tanto las imágenes resultantes como sus histogramas. Se emplean operaciones de recorte (slicing) para disminuir la resolución y se utiliza Matplotlib para ver de los resultados.

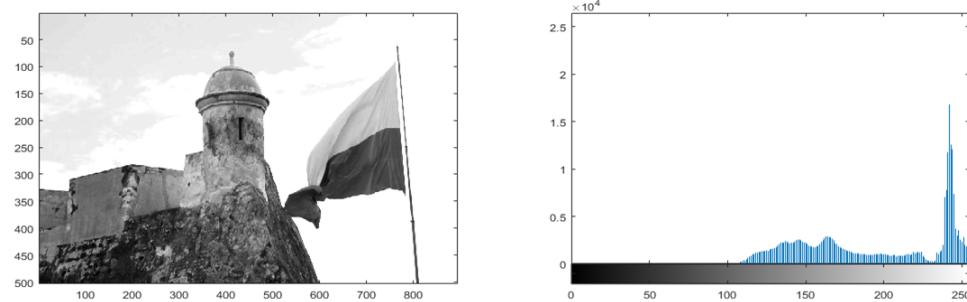
Solución

```
1 #Importación de Librería Básicas
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import sys
5 from ip_functions import *
6
7 # Abrir imagen
8 RGB=plt.imread("cartagena.jpg")
9 RGB=np.array(RGB)
10 #Realizar Slicing para reducir procesamiento
11 S=5
12
13 RGB=np.array(RGB[1::S,1::S,:])
14 #Seleccionar la capas
15 r=RGB[:, :, 0]
16 g=RGB[:, :, 1]
17 b=RGB[:, :, 2]
18
19 plt.figure(figsize=(15,15))
20 plt.subplot(2,3,1)
21 plt.imshow(r, cmap='gray')
22 plt.axis('off')
23 plt.title("Rojo")
24
25 plt.subplot(2,3,2)
26 plt.imshow(g, cmap='gray')
27 plt.axis('off')
28 plt.title("Verde")
29
30 plt.subplot(2,3,3)
31 plt.imshow(b, cmap='gray')
32 plt.axis('off')
33 plt.title("Azul")
34
35 ax4 = plt.subplot(2,3,4)
36 imhist(r, ax=ax4)
37
38 ax5 = plt.subplot(2,3,5)
39 imhist(g, ax=ax5)
40
41 ax6 = plt.subplot(2,3,6)
42 imhist(b, ax=ax6)
43
44 plt.tight_layout()
45 plt.show()
```

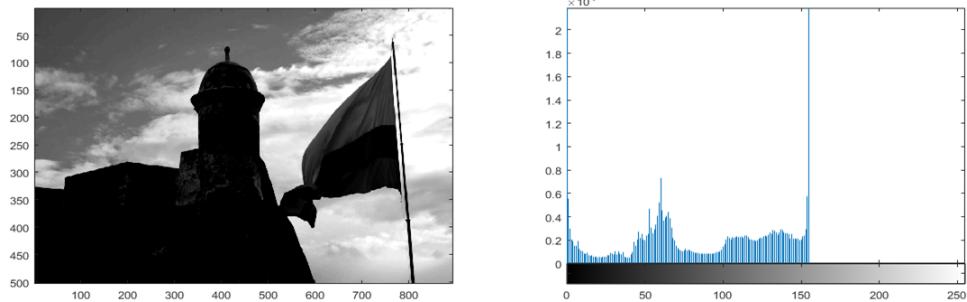
La Fig.4.5 muestra la separación de los canales del color RGB de la imagen de 'cartagen.jpg' y sus respectivos histogramas en Python. Las funciones referidas allí, se encuentran en el capítulo 13.



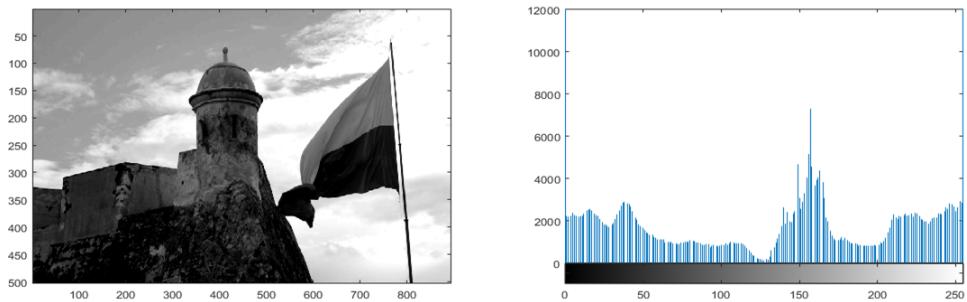
(a) Imagen de cartagena.jpg con brillo y contraste adecuado.



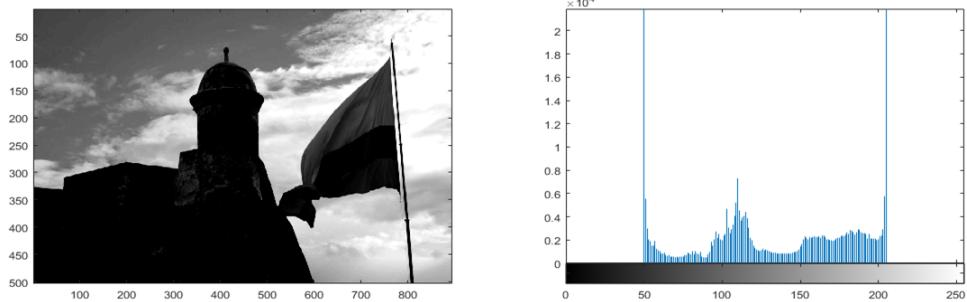
(b) Imagen de cartagena.jpg con alto brillo y bajo contraste.



(c) Imagen de cartagena.jpg con bajo brillo y bajo contraste.



(d) Imagen de cartagena.jpg con uniforme brillo y alto contraste.



(e) Imagen de cartagena.jpg con medio brillo y bajo contraste.

Figura 4.4: Ejemplos de análisis del histograma para evaluar brillo y contraste.

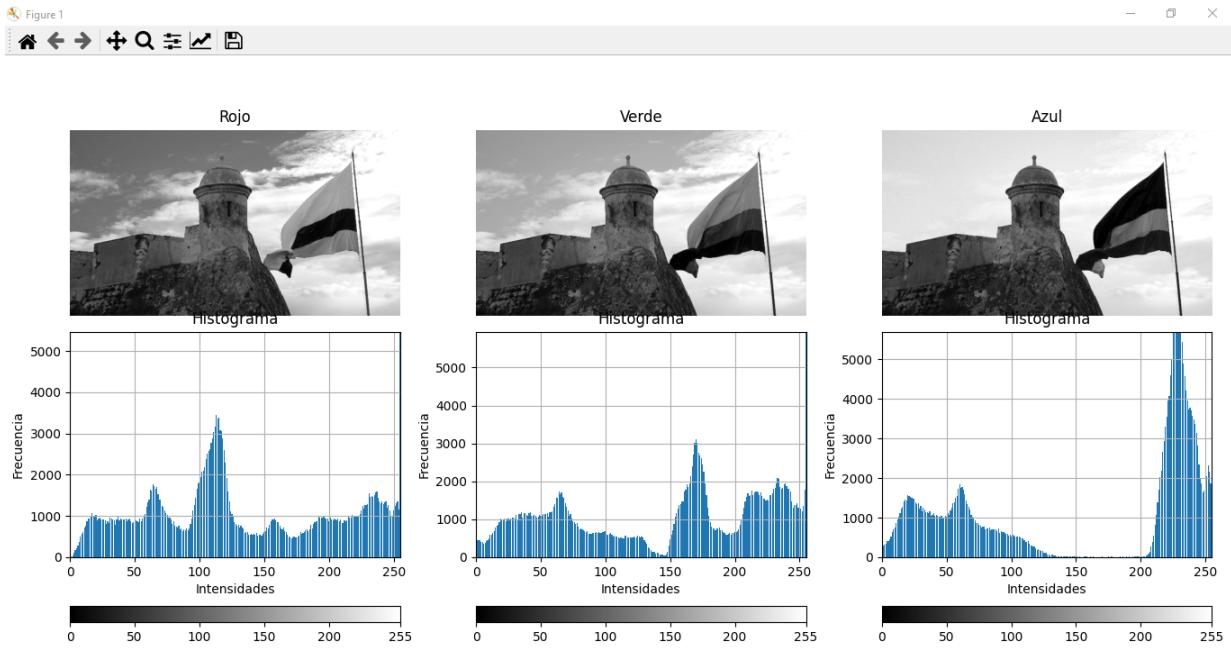


Figura 4.5: Separación de los canales del color RGB de la imagen y sus respectivos histogramas en Python.

4.1.2. El Brillo

El brillo se define como la cantidad de energía relativa emitida por una fuente de luz visible. En el contexto de las imágenes digitales, este término describe la intensidad lumínica percibida de los píxeles que conforman la imagen. A pesar de esta definición, es habitual emplear el término brillo para referirse a la intensidad con la que se presenta una imagen dentro de una escala determinada previamente que suele ir desde 0 hasta 255 en imágenes de 8 bits. Por lo anterior, una imagen con poco brillo tiene la mayoría de sus píxeles cercanos a 0 y una brillante próximos a 255 [5,13,28].



Figura 4.6: Ejemplos de imágenes con diferentes brillos. a) Bajo brillo, b) Alto brillo.

El brillo de una imagen I en escala de grises se puede determinar como la intensidad promedio de la capa respectiva como se muestra en (4.5):

$$B = \frac{1}{M \times N} \sum_{x=1}^M \sum_{y=1}^N I(x, y) \quad (4.5)$$

De igual forma, y de manera alternativa, el brillo también puede estimarse a partir del histograma $h(i)$ de ocurrencias o por histograma de probabilidad $p(i)$ como se presenta en (4.6) y (4.7) respectivamente:

$$B = \frac{\sum_{i=0}^{K-1} h(i) \times i}{\sum_{i=0}^{K-1} h(i)} \quad (4.6)$$

$$B = \sum_{i=0}^{K-1} p(i) \times i \quad (4.7)$$

4.1.3. El contraste

El contraste de una imagen se puede definir como diferencia relativa en intensidad entre una región de la imagen y su vecindad. Si ambas zonas tienen el mismo brillo, su diferencia relativa será cero y por lo tanto su contraste. Este depende de la percepción visual del observador, por lo que puede variar entre cada individuo. El contraste permite discriminar los objetos contenidos en una imagen de mejor manera. La Figura 4.7 entrega algunos ejemplos de imágenes donde se evidencia el cambio del contraste para un mismo objeto central el cual mantiene su brillo y solo varía su fondo [5,13,28].

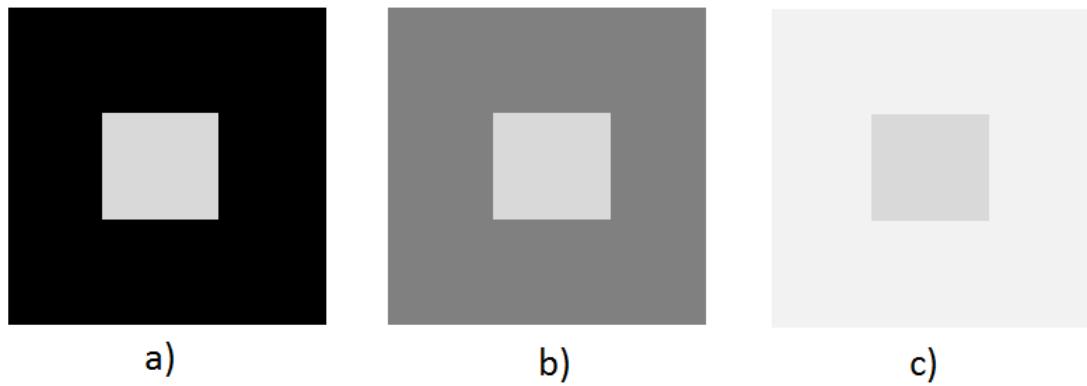


Figura 4.7: Ejemplos de imágenes con diferentes contrastes. a) Alto contraste, b) Mediano contraste y c) Bajo contraste.

Como ejemplo ilustrativo respecto a lo relativo que resulta el contraste, en la Figura 4.8 se muestra un claro ejemplo de dos objetos circulares , uno A y otro B, con igual intensidad sobre una amplia escala de grises. Al observador le puede parecer que las figuras A y B son de intensidades diferentes, pero en realidad son exactamente iguales. La razón por la que esto sucede es lo relativo del contraste al encontrarse la misma intensidad sobre diferentes fondos.



Figura 4.8: Dos figuras A y B sobre una amplia escala de grises. Aunque no resulta explícito, los tonos de grises son iguales en ambos lados.

El contraste de una imagen I en escala de grises se puede determinar como la desviación estándar de la capa respectiva como se muestra en (4.5):

$$C = \sqrt{\frac{1}{M \times N} \sum_{x=1}^M \sum_{y=1}^N [I(x, y) - B]^2} \quad (4.8)$$

De igual forma como se realizó con el brillo, y de manera alternativa, este también puede estimarse a partir del histograma $h(i)$ de ocurrencias o por histograma de probabilidad $p(i)$ como se presenta en (4.9) y (4.10), respectivamente:

$$C = \sqrt{\frac{\sum_{i=0}^{k-1} h(i) \times (i - B)^2}{\sum_{i=0}^{K-1} h(i)}} \quad (4.9)$$

$$C = \sqrt{\sum_{i=0}^{k-1} p(i) \times (i - B)^2} \quad (4.10)$$

4.1.4.

4.1.5. Ejemplo en clase: Construcción del histograma

Dada una imagen I en escala de grises de resolución 4×4 pixeles y una profundidad del color de 3 bits, determine:

$$I = \begin{bmatrix} 0 & 4 & 6 & 3 \\ 5 & 2 & 3 & 7 \\ 3 & 4 & 6 & 1 \\ 2 & 3 & 1 & 4 \end{bmatrix}$$

- a. El histograma de frecuencias.
- b. El histograma de probabilidades.
- c. El brillo y contraste a partir de los pixeles.
- d. El brillo y contraste a partir del histograma.

Solución:

- a. Para el histograma de frecuencias se procede a contar cuantos pixeles hay de cada una de las posibles clases. Si la imagen es de 3 bits, se generan $2^3 = 8$ clases que van desde $[0 - 7]$, como se indica en la segunda columna de la Tabla 4.1.
- b. Para el histograma de probabilidad $p(i)$, solo corresponde dividir el histograma de frecuencias $h(i)$ por la totalidad de pixeles de la imagen $4 \times 4 = 16$, como se muestra en la tercera columna de la Tabla 4.1.

Clase	Frecuencia $h(i)$	Probabilidad $p(i)$
0	1	$1/16=0.0625$
1	2	$2/16=0.1250$
2	2	$2/16=0.1250$
3	4	$4/16=0.2500$
4	3	$3/16=0.1875$
5	1	$1/16=0.0625$
6	2	$2/16=0.1250$
7	1	$1/16=0.0625$

Cuadro 4.1: Cuadro de los datos de los histogramas de frecuencia y probabilidad del ejemplo propuesto.

La Figura 4.9 presenta los respectivos histogramas del ejemplo enunciado. El eje de las abscisas corresponde a las clases y el de las ordenadas a la frecuencia de ocurrencia $h(i)$ y la probabilidad $p(i)$.

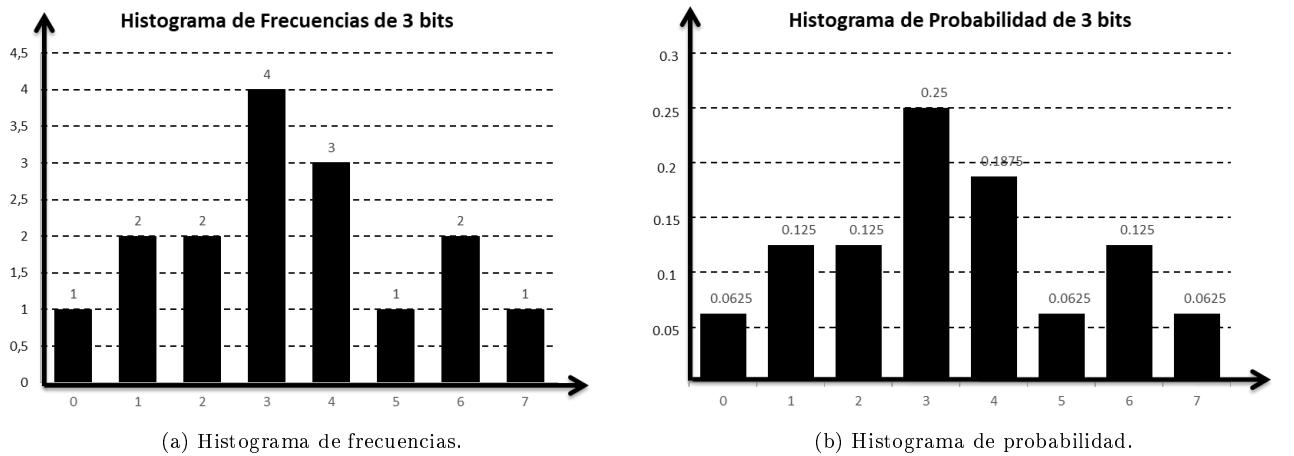


Figura 4.9: Histogramas para la imagen con $K = 2^3 = 8$ posibles valores de intensidad.

c. Para estimar el brillo a partir de los pixels se usa 4.11:

$$B = \frac{1}{N \times M} \sum_{x=1}^M \sum_{y=1}^N I(x, y) \quad (4.11)$$

$$B = \frac{0 + 4 + 6 + 3 + 5 + 2 + 3 + 7 + 3 + 4 + 6 + 1 + 2 + 3 + 1 + 4}{4 \times 4} = \frac{54}{16} = 3.375$$

y, finalmente, para calcular la el contraste desde la información de los píxeles se toma 4.12:

$$C = \sqrt{\frac{1}{N \times M} \sum_{x=1}^M \sum_{y=1}^N [I(x, y) - B]^2} \quad (4.12)$$

$$C = \left[\frac{1}{4 \times 4} \{ (0 - 3.375)^2 + (4 - 3.375)^2 + (6 - 3.375)^2 + (3 - 3.375)^2 + \dots \right.$$

$$\dots + (2 - 3.375)^2 + (3 - 3.375)^2 + (1 - 3.375)^2 + (4 - 3.375)^2 \} \right]^{\frac{1}{2}} = \sqrt{\frac{57.75}{16}}$$

$$C = 1.8998$$

d. Para la estimación del brillo se puede hacer por las frecuencias del histograma:

$$B = \frac{\sum_{i=0}^{K-1} h(i) \times i}{\sum_{i=0}^{K-1} h(i)}$$

$$B = \frac{0 \times 1 + 1 \times 2 + 2 \times 2 + 3 \times 4 + 4 \times 3 + 5 \times 1 + 6 \times 2 + 7 \times 1}{1 + 2 + 2 + 4 + 3 + 1 + 2 + 1} = \frac{54}{16}$$

$$B = 3.375$$

o por la probabilidad calculada a partir de histograma:

$$B = \sum_{i=0}^{K-1} p(i) \times i$$

$$B = 0 \times 0.0625 + 1 \times 0.125 + 2 \times 0.125 + 3 \times 0.25 + \dots$$

$$\dots + 4 \times 0.1875 + 5 \times 0.0625 + 6 \times 0.125 + 7 \times 0.0625$$

$$B = 3.375$$

De forma similar, para la estimación del contraste se puede realizar por frecuencias, tal y como se muestra en (4.13):

$$C = \sqrt{\frac{\sum_{i=0}^{K-1} h(i) \times (i - B)^2}{\sum_{i=0}^{K-1} h(i)}} \quad (4.13)$$

como:

$$\sum_{i=0}^{K-1} h(i) = 1 + 2 + 2 + 4 + 3 + 1 + 2 + 1 = 16$$

entonces

$$C = \left[\frac{1}{16} \{ (1 \times (0 - 3.375)^2 + 2 \times (1 - 3.375)^2 + 2 \times (2 - 3.375)^2 + 4 \times (3 - 3.375)^2 + \dots \right.$$

$$\dots + 3 \times (4 - 3.375)^2 + 1 \times (5 - 3.375)^2 + 2 \times (6 - 3.375)^2 + 1 \times (7 - 3.375)^2 \}]^{\frac{1}{2}} = \sqrt{\frac{57.75}{16}}$$

$$C = 1.8998$$

o por probabilidad se retoma 4.14:

$$C = \sqrt{\sum_{i=0}^{k-1} p(i) \times (i - B)^2} \quad (4.14)$$

$$C = [0.0625 \times (0 - 3.375)^2 + 0.125 \times (1 - 3.375)^2 + 0.125 \times (2 - 3.375)^2 + 0.25 \times (3 - 3.375)^2 + \dots$$

$$\dots + 0.1875 \times (4 - 3.375)^2 + 0.0652 \times (5 - 3.375)^2 + 0.125 \times (6 - 3.375)^2 + 0.0625 \times (7 - 3.375)^2]^{\frac{1}{2}}$$

$$C = 1.8998$$

Como puede apreciarse, los resultados son consistentes por ambos métodos.

4.2. Normalización del Histograma

El estiramiento del contraste, a menudo llamado normalización, es una técnica simple que busca mejorar el contraste general en una imagen “alargando” el rango de valores de intensidad que contiene la imagen de tal forma que se abarque un nuevo intervalo deseado de valores, el cual usualmente es llevado hasta el máximo permitido. Esta técnica acepta una imagen en escala de grises como entrada y producen otra imagen similar de salida, pero con un histograma modificado [5, 13].

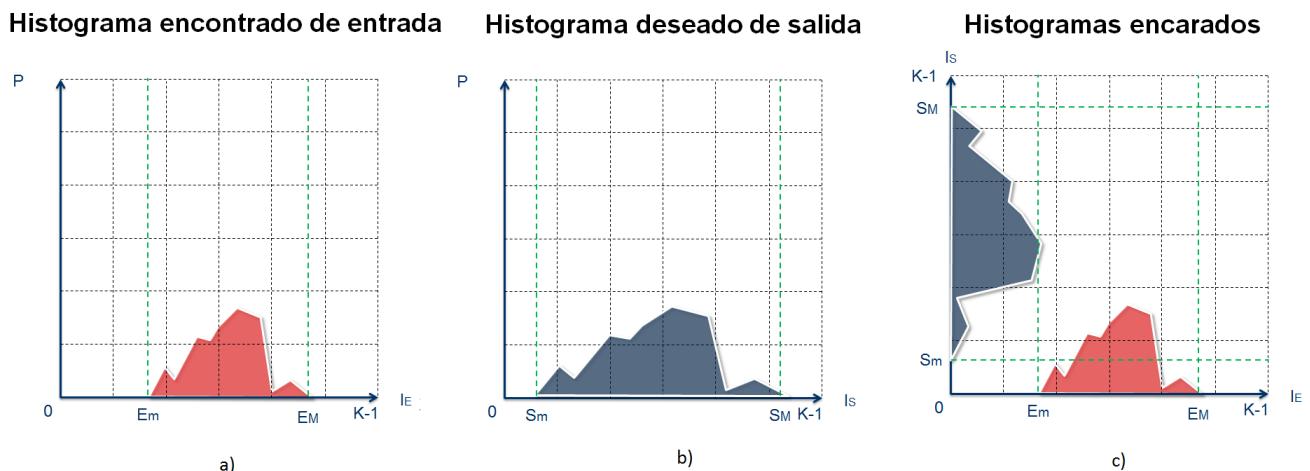


Figura 4.10: a) Histograma de entrada y su correspondiente. b) Histograma de salida deseado. c) Histogramas encarados

El ajuste de un histograma es realmente la ampliación o reducción del contraste de la imagen de entrada I_E . Esto se hace mediante la reasignación de las intensidades de los pixeles I_E de entrada a unos nuevos valores de salida I_s usando una curva para su mapeo (Véase la Figura 4.11).

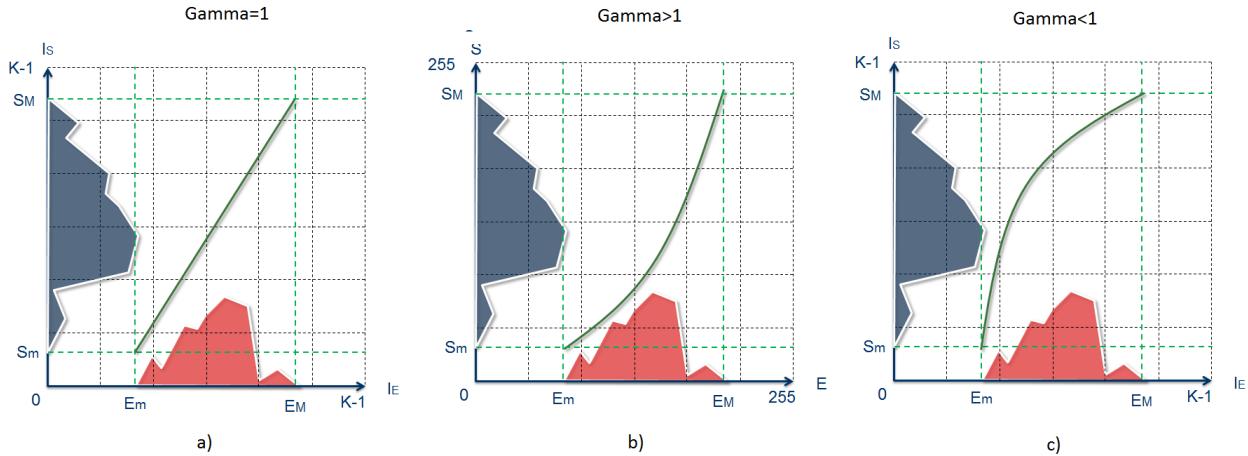


Figura 4.11: Comportamiento del gamma. a) Gamma lineal. b) Gamma mayor que uno. c) Gamma menor que uno y mayor que cero.

4.2.1. Ajuste por Potencia del Histograma

Se asume que E_m y E_M son los valores más bajo y más alto encontrados en el histograma de la imagen de entrada I_E , y S_m y S_M son los valores más bajo y más alto deseados del histograma de la imagen de salida I_S . Para lograr el ajuste, se puede realizar la transferencia a través de una curva potencia de la forma:

$$I_S = c I_E^n \quad (4.15)$$

Como la curva se inicia en (E_m, S_m) , entonces en (4.15) se transforma en:

$$I_S - S_m = c(I_E - E_m)^n \quad (4.16)$$

Con el objetivo de determinar la constante c , se puede usar una condición inicial tal como $I_S(E_M) = S_M$, así:

$$S_M - S_m = c(E_M - E_m)^n$$

$$c = \frac{(S_M - S_m)}{(E_M - E_m)^n} \quad (4.17)$$

Finalmente, evaluando (4.17) en (4.16) se obtiene la función que permite encontrar la equivalencia entre los píxeles de histograma de la imagen de entrada I_E y la imagen de salida ajustada I_S . La función de asignación para la operación de ajuste del contraste se define mediante (4.18):

$$I_S = \frac{S_M - S_m}{(E_M - E_m)^n} (I_E - E_m)^n + S_m \quad (4.18)$$

Donde:

I_E la matriz de intensidades de la imagen de entrada.

I_S la matriz de intensidades de la imagen de salida.

E_m es el valor escalar más bajo en el histograma de la imagen de entrada.

E_M es el valor escalar más alto en el histograma de la imagen de entrada.

S_m es el valor escalar más bajo deseado en el histograma de la imagen de salida.

S_M es el valor escalar más alto deseado en el histograma de la imagen de salida.

n es el gamma deseado para la imagen de salida.

El rango objetivo $[S_m, S_M]$ no es necesariamente el máximo de valores disponibles dado por el número de bits de la imagen, por lo que puede ser cualquier intervalo para el que la imagen de entrada I_E se desea proyectar, por lo que el método también se puede utilizar para reducir el contraste de salida. Esto es posible siempre que $E_m \neq E_M$; es decir, que la imagen contenga al menos *dos* valores de pixeles diferentes.

Para una imagen de 8 bits con $S_m = 0$, $S_M = 255$ y $n = 1$, la función en (4.18) se simplifica a (4.19). Véase la Figura 4.12.

$$I_S = \frac{255}{(E_M - E_m)}(I_E - E_m) + S_m \quad (4.19)$$

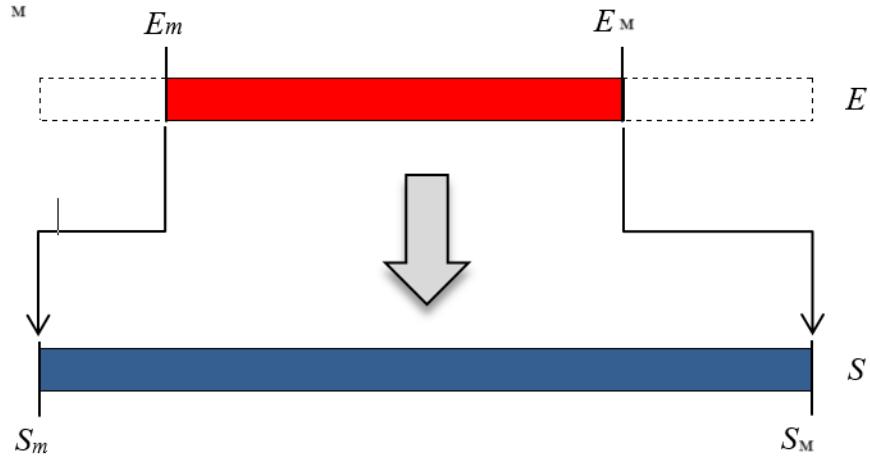


Figura 4.12: Operación de ajuste de contraste. .

La Gamma

En fotografía, la gamma se refiere a la relación entre las luces y las sombras en una imagen. Es una medida de la cantidad de cambio de brillo que ocurre cuando se ajusta la exposición de una imagen. Esta se expresa como un número que representa la relación entre la intensidad de la luz de entrada y la intensidad de la luz de salida en una imagen. Un valor de gamma de 1 indica que la relación es lineal, lo que significa que un cambio en la intensidad de entrada se traduce en un cambio proporcional en la intensidad de salida. Por otro lado, el valor mayor que 1 expresa una relación no lineal, donde un cambio en la intensidad de entrada produce un cambio desproporcionado en la

intensidad de salida. En términos de exposición, un ajuste de gamma se puede utilizar para mejorar la apariencia general de una imagen. Si una imagen tiene demasiado contraste entre las luces y las sombras, un ajuste de gamma puede suavizar la transición entre ellas, lo que resulta en una versión más equilibrada y fácil de ver. Por otro lado, si la imagen tiene un contraste pobre, un ajuste de gamma puede incrementar la diferencia entre las luces y las sombras, mejorando así la percepción visual. A manera de ilustración, se muestra la Figura 4.13 que correspondiente al Castillo San Felipe en Cartagena (Colombia), donde en la parte superior se ve el resultado del la transformación y en la inferior su histograma. a) y b) con $\text{Gamma}=0.5$ con favorecimiento de los tonos claros sobre los oscuros, c) y d) con $\text{Gamma}=1$ con la misma distribución de la tonalidades, y e) y f) con $\text{Gamma}=1.5$ dando ventaja a las tonalidades oscuras sobre las claras [5, 13].

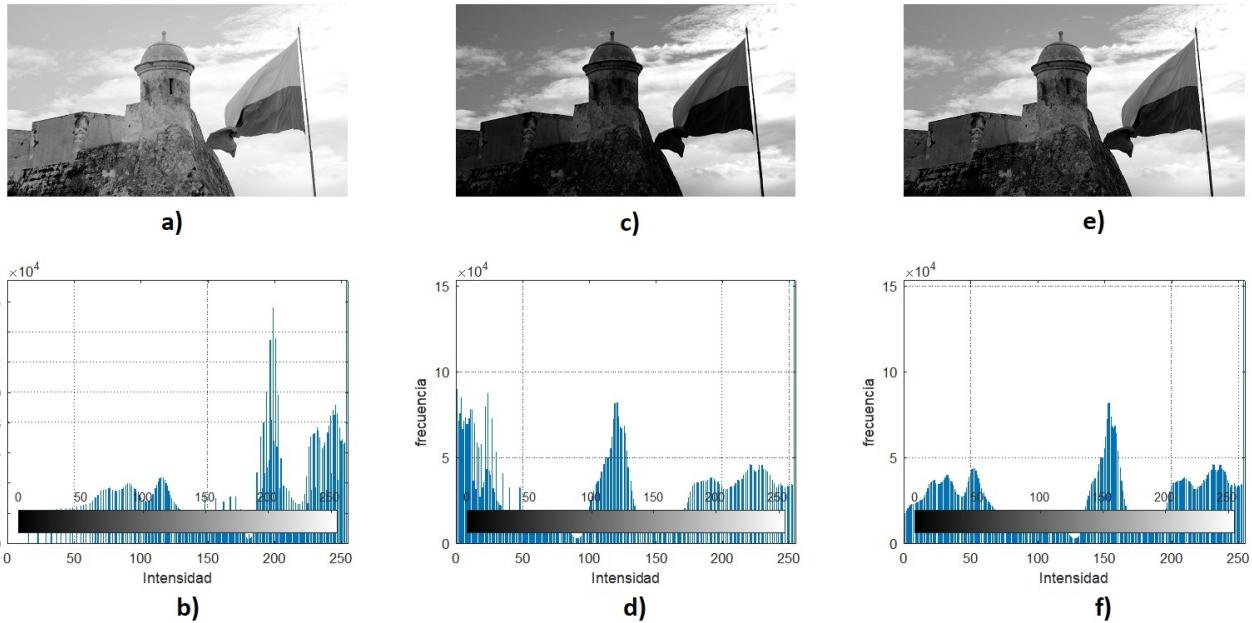


Figura 4.13: Imagen de Cartagena con diferentes ajustes de Gamma. En la parte superior su resultado y en la inferior su histograma. a) y b) con $\text{Gamma}=0.5$, c) y d) con $\text{Gamma}=1$, y e) y f) con $\text{Gamma}=1.5$.

4.2.2. Ejemplo en clase: ajuste del histograma por Potencia

Dada una imagen I_E de entrada en escala de grises de resolución 4×4 pixeles y una profundidad del color de 3 bits, determine el histograma usando gamma igual a 1, la imagen ajustada y su nuevo histograma.

4	3	3	5
6	2	2	5
2	3	5	6
4	2	2	5

Figura 4.14: Imagen de 3 bits.

Solución:

En la Figura 4.15 se muestra el histograma sin ajustar.

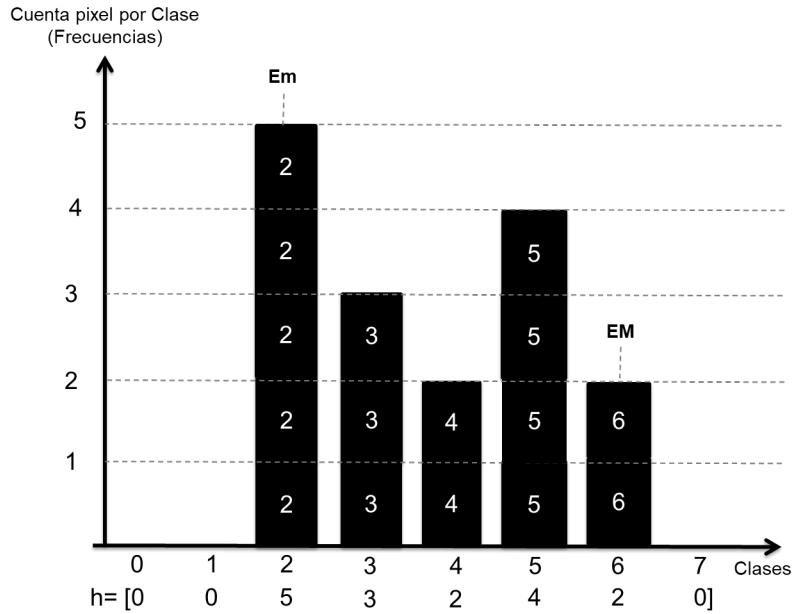


Figura 4.15: Histograma de frecuencias.

Las características del histograma son:

Valor más bajo, $Em = 2$

Valor más alto, $EM = 6$

Valor más bajo deseado, $S_m = 0$

Valor más alto deseado, $SM = 7$

Gama $n = 1$

Usando (4.20), se obtienen los valores I_S de salida para cada entrada I_E , como se muestra en la Tabla 4.2.

$$I_S = \frac{S_M - S_m}{(E_M - E_m)^n} (I_E - E_m)^n + S_m \quad (4.20)$$

Intensidad I_E	Cálculo de Salida I_S	Intensidad I_S	I_S redondeado
2	$\frac{(7-0)}{(6-2)} [2 - 2] + 0 = 0$	0.00	0
3	$\frac{(7-0)}{(6-2)} [3 - 2] + 0$	1.75	2
4	$\frac{(7-0)}{(6-2)} [4 - 2] + 0$	3.50	4
5	$\frac{(7-0)}{(6-2)} [5 - 2] + 0$	5.25	5
6	$\frac{(7-0)}{(6-2)} [6 - 2] + 0$	7.00	7

Cuadro 4.2: Cálculo de la intensidad de Salida I_S a partir de la intensidad de entrada I_E . Nótese el redondeo al entero más cercano.

Para obtener una matriz ajustada de la imagen original de 3 bits, se reemplaza el valor de intensidad I_E en Figura 4.14, por su respectivo resultado de I_S aproximado. Finalmente, la imagen ajustada se muestra en la Figura 4.16 y su nuevo histograma en Figura 4.17.

4	2	2	5
7	0	0	5
0	2	5	7
4	0	0	5

Figura 4.16: Imagen de 3 bits ajustada.

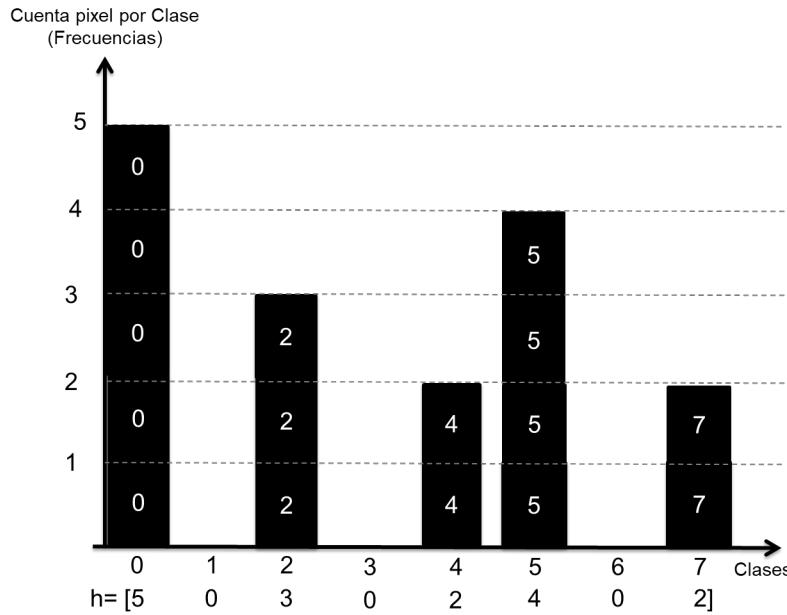


Figura 4.17: Histograma de frecuencias ajustado.

4.2.3. Ajuste Exponencial del Histograma

La función exponencial se utiliza para aumentar el contraste de las zonas claras en detrimento de las oscuras. Nuevamente se asume que E_m y E_M son los valores más bajo y más alto encontrados en el histograma de la imagen de entrada I_E , y S_m y S_M son los valores más bajo y más alto deseados del histograma de la imagen de salida I_S . Para lograr el ajuste, se puede realizar la transferencia a través de una curva exponencial de la forma mostrada en 4.21. Debe destacarse que para este caso, los valores de las intensidades deben normalizarse entre 0 y 1 debido a que la función exponencia crece muy rápido. La Figura 4.18 ilustra el resultado de ajustar la imagen de Cartagena bajo este criterio.

$$I_S = \frac{(S_M - S_m)}{\left(e^{(E_M - E_m)} - 1\right)} \left[e^{(I_E - E_m)} - 1 \right] + S_m \quad (4.21)$$

4.2.4. Ajuste Logarítmico del Histograma

Igual que en los casos potencia y exponencial, aquí se asume que E_m y E_M son los valores más bajo y más alto encontrados en el histograma de la imagen de entrada I_E , y S_m y S_M son los valores más bajo y más alto deseados del histograma de la imagen de salida I_S . Para realizar el cambio, se hace el mapeo a través de una curva logarítmica natural de la forma mostrada en (4.22). Debe destacarse que para este caso también, los valores de las intensidades

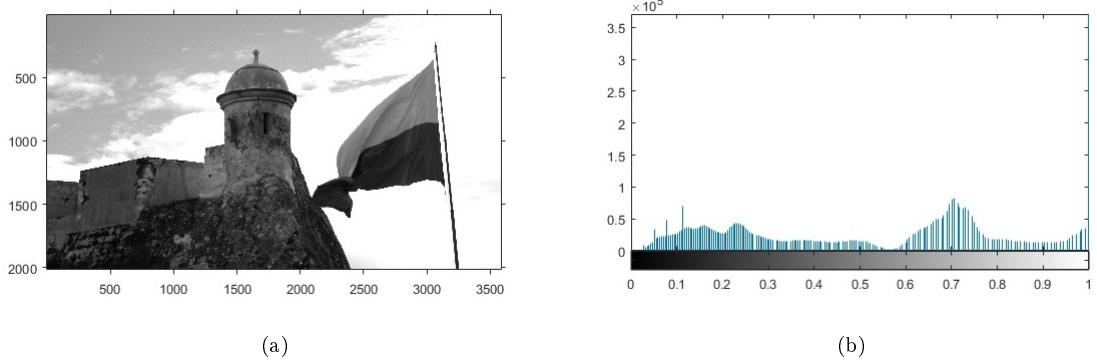


Figura 4.18: Ajuste de la imagen de 'cartagena.jpg' usando función exponencial. (a) Imagen ajustada y (b) Histograma ajustado.

tiene que estar normalizadas entre 0 y 1 debido a que la función potencia cambia rápidamente. La Figura 4.19 ilustra el resultado de ajustar la imagen de Cartagena bajo este criterio.

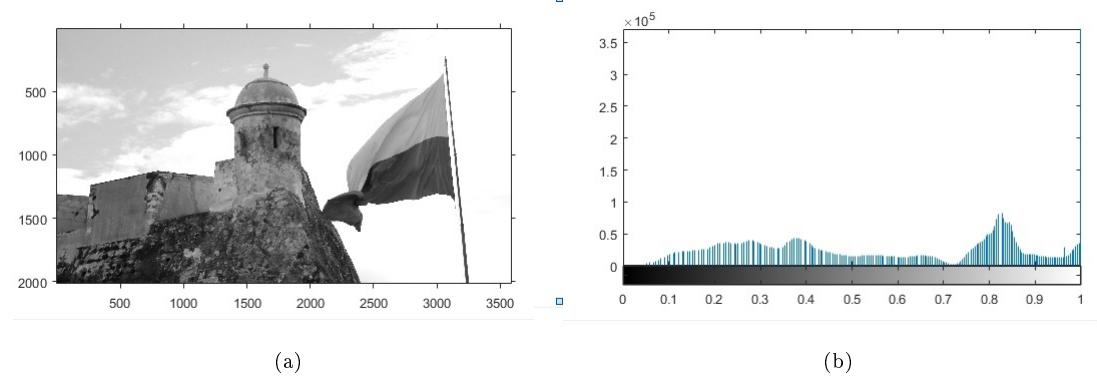


Figura 4.19: Ajuste de la imagen de 'cartagena.jpg' usando función logaritmo. (a) Imagen ajustada y (b) Histograma ajustado.

$$I_S = \frac{(S_m - S_m)}{\ln(E_M - E_m + 1)} \ln[I_E - E_m + 1] + S_m \quad (4.22)$$

4.2.5. Determinación de los Valores E_m y E_M

Si bien visualmente se pueden estimar los valores E_m y E_M a partir del histograma como los límites mínimo y máximo donde se encuentra el cuerpo principal de los modos (cúmulos), es algo que no resulta práctico cuando se desea realizar de forma automática el ajuste. Un criterio que usualmente se aplica es considerar el 1% del área bajo la curva en ambos lados del histograma como punto para la selección de la intensidades límites E_m y E_M , no obstante se pueden modificar el porcentaje manualmente a cada necesidad. Véase la Figura 4.20.

Para ajustar el contraste, se determinan los dos límites en entrada: inferior E_m y superior E_M . Estos se encuentran explorando fácilmente el histograma acumulado de probabilidad $H_p(i)$ para identificar los índices (niveles de intensidad) en los que la suma alcanza porcentajes predefinidos como fronteras. Típicamente, el límite inferior es el valor de intensidad E_m más bajo para el cual el histograma acumulado es mayor o igual a un pequeño porcentaje (por ejemplo, 1), mientras que el límite superior E_M es el valor más alto para el cual el acumulado es menor o igual al complemento de ese porcentaje (por ejemplo, 99). El ajuste de contraste y la determinación asertiva de los límites asegura que se preserven los detalles en las áreas oscuras y brillantes, mientras se mejora el rango dinámico general

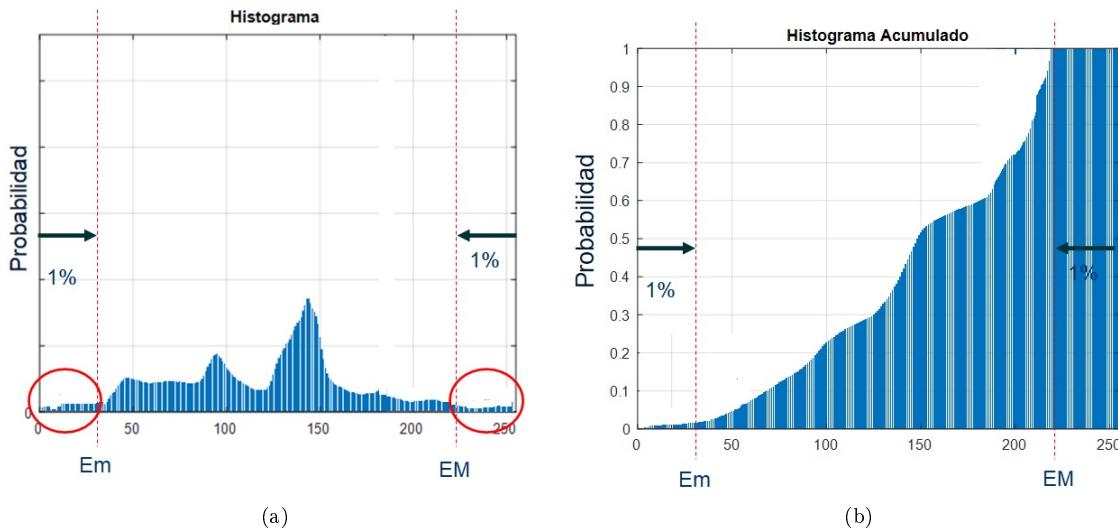


Figura 4.20: Criterios de selección de los valores de Em y EM. En (a) se muestra el histograma de probabilidad con los límites y en (b) se aprecian las áreas bajo la curva del histograma acumulado.

de la imagen.

4.2.6. Ejemplo en Python: Ajuste de una imagen

Este código en Python ilustra el procesamiento de tres fotografías tomadas con filtros de color rojo, verde y azul, (acrílicos transparentes de colores o papel celofán) utilizando una cámara en modo blanco y negro, junto con una referencia en color tomada sin mover en ningún caso la cámara. Las imágenes filtradas se cargan y procesan para formar sus respectivas capas de color en un modelo RGB. Posteriormente, se realiza un ajuste de histograma en cada capa utilizando los valores mínimos y máximos de entrada y salida mediante las funciones. Finalmente, las tres capas ajustadas se combinan para reconstruir la imagen a color, la cual se compara con la referencia original y con la versión no ajustada. Los resultados incluyen la visualización de las capas, los histogramas y las imágenes combinadas, como se aprecia en la figura.

Solución

```

1 #Importación de Librería Básicas
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import sys
5 from ip_functions import *
6
7 # Abrir Imagen
8 Color=plt.imread("ColorA.jpg")
9 R=plt.imread("RojoA.jpg")
10 G=plt.imread("VerdeA.jpg")
11 B=plt.imread("AzulA.jpg")
12
13 #Realizar Slicing para reducir procesamiento
14 S=10
15 Color=np.array(Color[1::S,1::S,:])
16 R=np.array(R[1::S,1::S,:])
17 G=np.array(G[1::S,1::S,:])
18 B=np.array(B[1::S,1::S,:])
19
20 #Seleccionar la capa Roja Positiva
21 r=np.array(R[:, :, 0])
22 g=np.array(G[:, :, 1])
23 b=np.array(B[:, :, 2])
24
25 #Mostrar las Capas

```

```

26 plt.figure(figsize=(15,15))
27 plt.subplot(2,3,1)
28 plt.imshow(r,cmap='gray')
29 plt.axis('off')
30 plt.title("Rojo")
31 plt.subplot(2,3,2)
32 plt.imshow(g,cmap='gray')
33 plt.axis('off')
34 plt.title("Verde")
35 plt.subplot(2,3,3)
36 plt.imshow(b,cmap='gray')
37 plt.axis('off')
38 plt.title("Azul")
39
40 ax4=plt.subplot(2,3,4)
41 imhist(r,ax=ax4)
42 plt.title("Rojo")
43 ax5=plt.subplot(2,3,5)
44 imhist(g,ax=ax5)
45 plt.title("Verde")
46 ax6=plt.subplot(2,3,6)
47 imhist(r,ax=ax6)
48 plt.title("Azul")
49
50 #Ajuste del Histograma
51 #Capa Roja
52 #Emr=25/255
53 #EMr=80/255
54 Emr,EMr=stretchlim(r);
55 Smr=0/255
56 SMr=255/255
57
58 n=1 #El mismo para todas las capas
59 #Capa Verde
60 #Emg=25/255
61 #EMg=140/255
62 Emg,EMg=stretchlim(g);
63 Smg=0/255
64 SMg=255/255
65
66 #Capa VerAzulde
67 #Emb=25/255
68 #EMb=80/255
69 Emb,EMb=stretchlim(b);
70 Smb=0/255
71 SMb=255/255
72
73 Isr=imadjust(r,[Emr,EMr],[Smr,SMr],n)
74 Isg=imadjust(g,[Emg,EMg],[Smg,SMg],n)
75 Isb=imadjust(b,[Emb,EMb],[Smb,SMb],n)
76
77 #Gráficas Ajustadas de Salida
78 plt.figure(figsize=(15,15))
79 plt.subplot(2,3,1)
80 plt.imshow(Isr,cmap='gray')
81 plt.axis('off')
82 plt.title("Rojo")
83 plt.subplot(2,3,2)
84 plt.imshow(Isg,cmap='gray')
85 plt.axis('off')
86 plt.title("Verde")
87 plt.subplot(2,3,3)
88 plt.imshow(Isb,cmap='gray')
89 plt.axis('off')
90 plt.title("Azul")
91
92 ax4=plt.subplot(2,3,4)
93 imhist(Isr,ax=ax4)
94 plt.title("Rojo")
95 ax5=plt.subplot(2,3,5)
96 imhist(Isg,ax=ax5)

```

```

97 plt.title("Verde")
98 ax6=plt.subplot(2,3,6)
99 imhist(Isb,ax=ax6)
100 plt.title("Azul")
101
102 #Histogramas Ajustados de Salida
103 #Imágenes de Referencia, Compuesta y Ajusta
104 In=np.uint8(np.stack((r,g,b),axis=-1))
105 Ia=np.uint8(np.stack((Isr,Isg,Isb),axis=-1))
106 plt.figure(figsize=(18,15))
107 plt.subplot(1,3,1)
108 plt.imshow(Color)
109 plt.title('Referencia')
110 plt.subplot(1,3,2)
111 plt.imshow(In)
112 plt.title('Compuesta')
113 plt.subplot(1,3,3)
114 plt.imshow(Ia)
115 plt.title('Ajustada')
116 plt.axis('off')
117 plt.show()

```

La Fig.4.21 muestra el proceso de composición de una imagen a color la cual se generó a partir de filtros de color con la cámara en modo escala de grises. Los valores de los mínimos y máximos para el ajuste de los histogramas se pueden seleccionar a partir de los histogramas previos de manera manual o automática. Las funciones referidas allí, se encuentran en el capítulo 13.

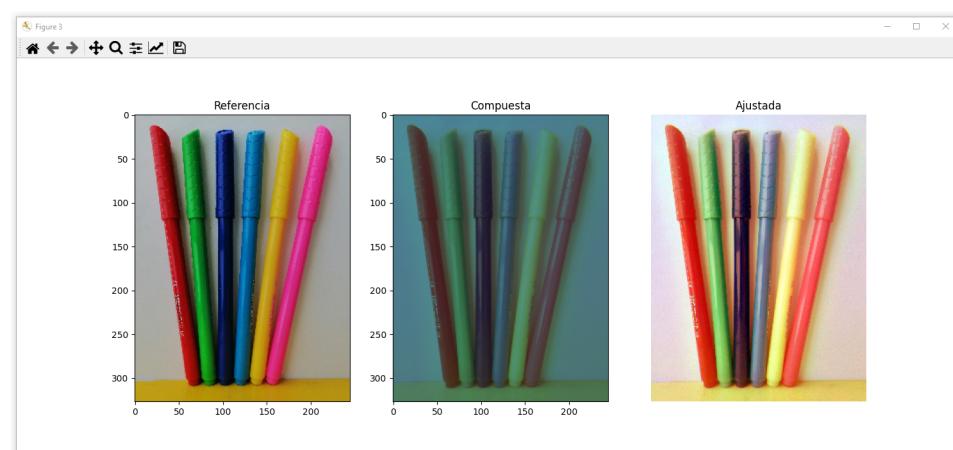
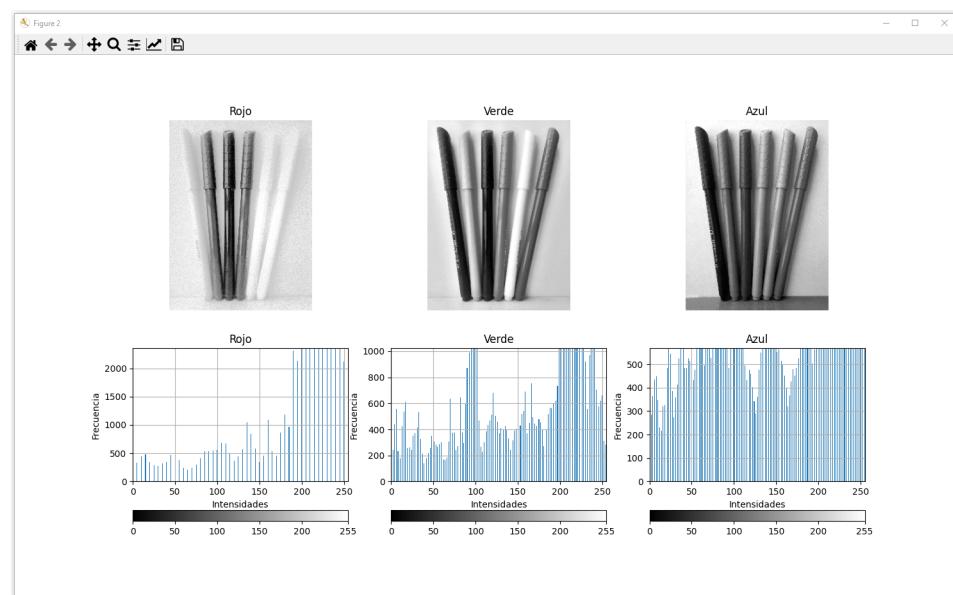
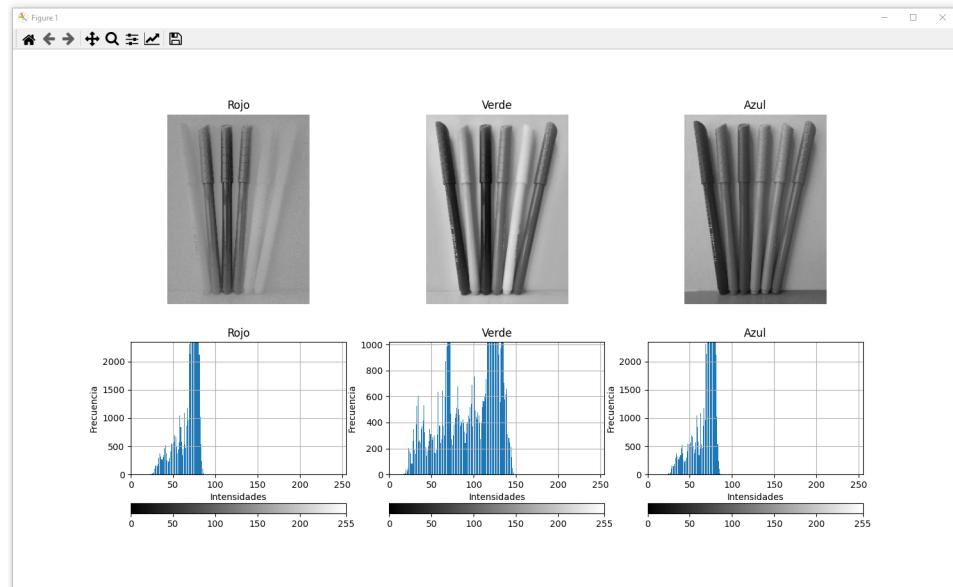


Figura 4.21: La parte superior ilustra los histogramas de las fotografías tomadas en escala de grises, el la central se aprecia una vez se han ajustado las intensidades y, en la parte inferior de izquierda a derecha, la imagen de referencia, la compuesta sin ajustar y, finalmente, la ajustada.

4.3. Ecualización del histograma

La ecualización es una técnica de procesamiento de imágenes que se utiliza con frecuencia para obtener imágenes con mejor contraste en las aplicaciones médicas tales como radiografías, resonancias magnéticas y tomografías computarizadas. Todas estas requieren alta definición de bordes y una mejora del contraste del color para facilitar la determinación de una característica que se puede apreciar y así llegar a diagnosticar con un patrón una patología. Sin embargo, en algunos casos, la ecualización del histograma puede enfatizar también el ruido oculto en la imagen, por lo que usualmente se ejecuta acompañada de otras operaciones. El objetivo de la ecualización del histograma es redistribuir los valores de intensidad de los píxeles de una imagen de manera que el de salida sea lo más plano posible; es decir, procurando que todas las intensidades presentes en la imagen ocurran con la misma frecuencia de ocurrencia o probabilidad posible [5, 13, 33].

El proceso comienza calculando el histograma de la imagen de entrada $h(i)$, que representa la frecuencia de ocurrencia de cada nivel de intensidad. Luego, se construye el histograma acumulado de probabilidades $H(i)$ sumando secuencialmente las frecuencias normalizadas desde el nivel de intensidad más bajo hasta el más alto. A continuación, se determina la relación entre el histograma acumulado de probabilidades de la imagen de entrada y los niveles de intensidad deseados en la imagen ecualizada. La Figura 4.22 muestra conceptualmente el proceso para lograr la ecualización. En la parte a) se aprecia el histograma de entrada de probabilidad p_E y en la b) el histograma de probabilidad aplastado de salida p_S .

4.3.1. Ejemplo en clase: Ecualización

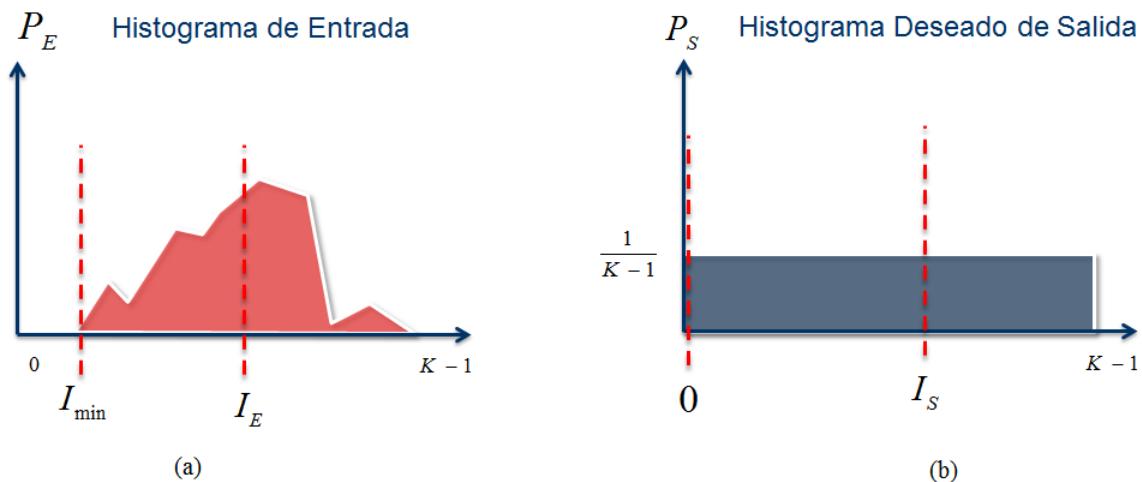


Figura 4.22: Modelo para la ecualización del histograma en su versión discreta. (a) Histograma de entrada de probabilidad y (b) Histograma ecualizado deseado de salida.

Para aproximarse a la ecualización de una imagen, se debe lograr que la probabilidad de ocurrencia del histograma de salida sea lo más plana posible, reconociendo que el área bajo ambos histogramas es la misma. Teniendo en cuenta lo anterior, se puede proponer (4.23) a partir de la Figura 4.22:

$$\sum_{i=I_{\min}}^{I_E} p_E(i) = \sum_{i=0}^{I_S} p_S(i) \quad (4.23)$$

El área bajo la curva del histograma de salida h_S hasta I_S se puede escribir como se muestra en (4.24):

$$\sum_{i=0}^{I_S} p_S(i) = \frac{1}{(K-1)} I_S \quad (4.24)$$

Si se reemplaza (4.25) en (4.23), se obtiene:

$$\sum_{i=I_{min}}^{I_E} p_E(i) = \frac{1}{(K-1)} I_S \quad (4.25)$$

Despejando I_S , se llega a (4.26):

$$I_S = (K-1) \sum_{i=I_{min}}^{I_E} p_E(i) \quad (4.26)$$

Al escribir en términos del histograma acumulado de probabilidad H_E hasta I_E :

$$H_E(I_E) = \sum_{i=I_{min}}^{I_E} p_E(i) \quad (4.27)$$

Así, evaluando (4.27) en (4.26) se logra la intensidad equivalente en el histograma ecualizado. Véase en (4.28):

$$I_S = (K-1) H_E(I_E) \quad (4.28)$$

4.3.2. Ejemplo en Clase: Ajuste por Ecualización

Dada una imagen I_E de entrada en escala de grises de resolución 4×4 pixeles y una profundidad del color de 3 bits, realice la ecualización del histograma y muestre la imagen (matriz) ecualizada de la entrada de la Figura 4.23.

4	3	3	5
6	2	2	5
2	3	5	6
4	2	2	5

Figura 4.23: Imagen de entrada 3 bits.

Solución:

Se muestra en la Figura 4.24 el histograma en entrada h .

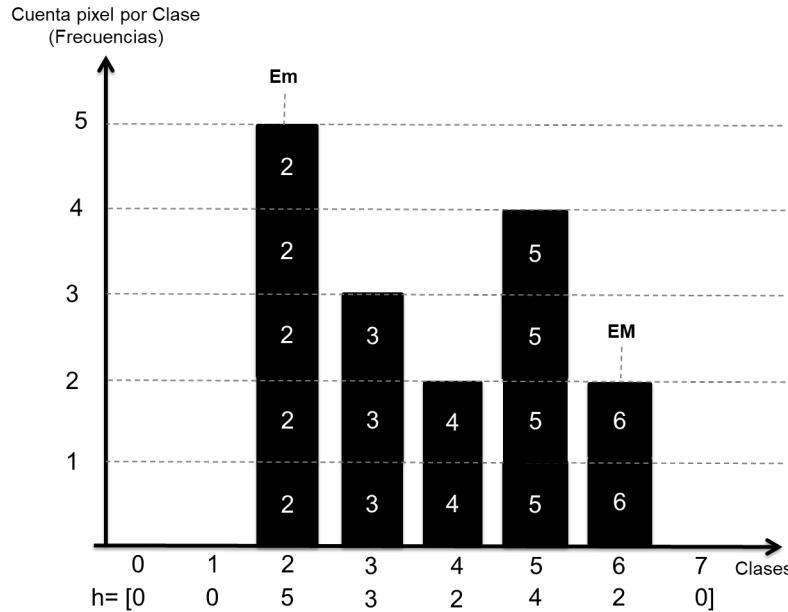


Figura 4.24: Histograma de frecuencias h .

Para realizar la ecualización se debe utilizar (4.29):

$$I_S = (K - 1) H_E(I_E) \quad (4.29)$$

A partir del histograma de la Figura 4.24, se construye la Tabla 4.3.

I_E	Frecuencia h	F Acumulada h_a	P Probabilidad	P Acumulada H_E	Intensidad I_S	I_S Redondeado
2	5	5	$5/16=0.3125$	0.3125	$7 \times 0.3125=2.16$	2
3	3	8	$3/16=0.1875$	0.5000	$7 \times 0.5000=3.50$	4
4	2	10	$2/16=0.1250$	0.6250	$7 \times 0.6250=4.50$	5
5	4	14	$4/16=0.2500$	0.8750	$7 \times 0.8705=6.10$	6
6	2	16	$2/16=0.1250$	1.0000	$7 \times 1.000=7.00$	7

Cuadro 4.3: Datos para el cálculo de la ecualización del histograma.

Con lo que se obtiene la imagen ecualizada mostrada en la Figura 4.25 y su respectivo histograma en la Figura 4.26.

5	4	4	6
7	2	2	6
2	4	6	7
5	2	2	6

Figura 4.25: Imagen ecualizada de 3 bits.

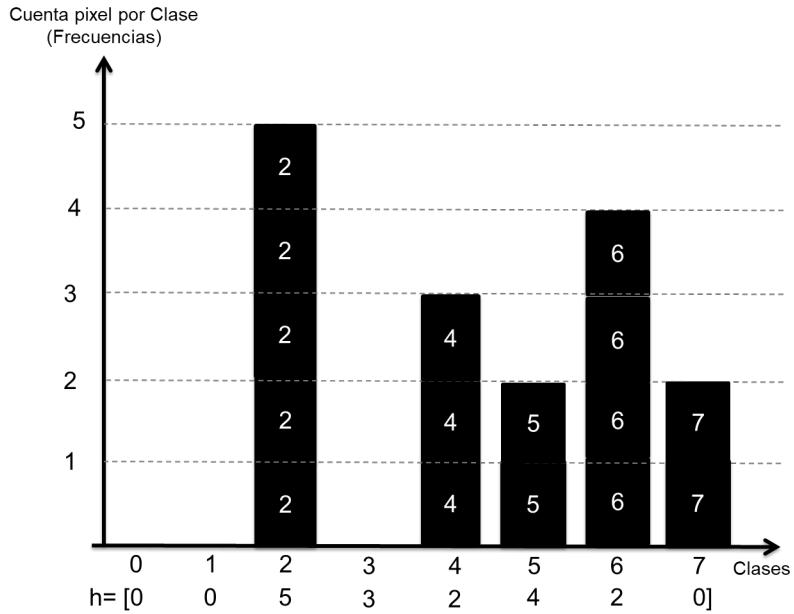


Figura 4.26: Histograma ecualizado de 3 bits.

4.3.3. Ejemplo en Python: Ecualización de una imagen

Este código en Python ilustra el procesamiento de tres fotografías tomadas con filtros de color rojo, verde y azul (acrílicos transparentes de colores o papel celofán) utilizando una cámara en modo blanco y negro, junto con una referencia en color tomada sin mover la cámara en ningún caso. Las imágenes filtradas se cargan y procesan para formar sus respectivas capas de color en un modelo RGB. Posteriormente, se realiza una ecualización de histograma en cada capa de manera automática. Este proceso no requiere valores mínimos ni máximos de entrada y salida para su operación, ya que redistribuye los niveles de intensidad en toda la escala de grises, aplana el histograma y mejora el contraste general de la imagen. Finalmente, las tres capas ecualizadas se combinan para reconstruir la imagen a color, la cual se compara con la referencia original y con la versión no ecualizada. Los resultados incluyen la visualización de las capas, los histogramas y las imágenes combinadas, como se aprecia en la figura.

Solución

```

1 #Importación de Librería Básicas
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import sys
5 from ip_functions import *
6
7 # Abrir Imagen
8 Color=plt.imread("ColorA.jpg")
9 R=plt.imread("RojoA.jpg")
10 G=plt.imread("VerdeA.jpg")
11 B=plt.imread("AzulA.jpg")
12
13 #Realizar Slicing para reducir procesamiento
14 S=10
15
16 Color=np.array(Color[1::S,1::S,:])
17 R=np.array(R[1::S,1::S,:])
18 G=np.array(G[1::S,1::S,:])
19 B=np.array(B[1::S,1::S,:])
20
21 #Seleccionar la capa Roja Positiva
22 r=np.array(R[:, :, 0])
23 g=np.array(G[:, :, 1])
24 b=np.array(B[:, :, 2])

```

```

25 # r=non_overflowing_sum(r,100)
26
27 """#Mostar las Capas"""
28 plt.figure(figsize=(15,15))
29 plt.subplot(2,3,1)
30 plt.imshow(r,cmap='gray')
31 plt.axis('off')
32 plt.title("Rojo")
33 plt.subplot(2,3,2)
34 plt.imshow(g,cmap='gray')
35 plt.axis('off')
36 plt.title("Verde")
37 plt.subplot(2,3,3)
38 plt.imshow(b,cmap='gray')
39 plt.axis('off')
40 plt.title("Azul")
41
42 ax4=plt.subplot(2,3,4)
43 imhist(r,ax=ax4)
44 plt.title("Rojo")
45 ax5=plt.subplot(2,3,5)
46 imhist(g,ax=ax5)
47 plt.title("Verde")
48 ax6=plt.subplot(2,3,6)
49 imhist(r,ax=ax6)
50 plt.title("Azul")
51
52 """#Ecualización del Histograma"""
53 #Capa Roja
54 Isr=histeq(r)
55 #Capa Verde
56 Isg=histeq(g)
57 #Capa Azul
58 Isb=histeq(b)
59
60 """#Gráficas Ajustadas de Salida"""
61 plt.figure(figsize=(15,15))
62 plt.subplot(2,3,1)
63 plt.imshow(Isr,cmap='gray')
64 plt.axis('off')
65 plt.title("Rojo")
66 plt.subplot(2,3,2)
67 plt.imshow(Isg,cmap='gray')
68 plt.axis('off')
69 plt.title("Verde")
70 plt.subplot(2,3,3)
71 plt.imshow(Isb,cmap='gray')
72 plt.axis('off')
73 plt.title("Azul")
74
75 ax4=plt.subplot(2,3,4)
76 imhist(Isr,ax=ax4)
77 plt.title("Rojo")
78 ax5=plt.subplot(2,3,5)
79 imhist(Isg,ax=ax5)
80 plt.title("Verde")
81 ax6=plt.subplot(2,3,6)
82 imhist(Isb,ax=ax6)
83 plt.title("Azul")
84
85 """Histogramas Ajustados de Salida
86 #Imágenes de Referencia, Compuesta y Ajusta
87 """
88 In=np.uint8(np.stack((r,g,b),axis=-1))
89 Ia=np.uint8(np.stack((Isr,Isg,Isb),axis=-1))
90 plt.figure(figsize=(18,15))
91 plt.subplot(1,3,1)
92 plt.imshow(Color)
93 plt.title('Referencia')
94 plt.subplot(1,3,2)
95 plt.imshow(In)

```

```
96 plt.title('Compuesta')
97 plt.subplot(1,3,3)
98 plt.imshow(Ia)
99 plt.title('Ecualizada')
100 plt.axis('off')
101 plt.show()
```

La Fig.[4.27](#) muestra el proceso de composición de una imagen a color la cual se generó a partir de filtros de color con la cámara en modo escala de grises. El algoritmo en Python realiza la ecualización de los histogramas que hacen se mejore el contraste, pero enfatizan el ruido en comparación a lo realizado por el ajuste. Las funciones referidas allí, se encuentran en el capítulo [13](#).

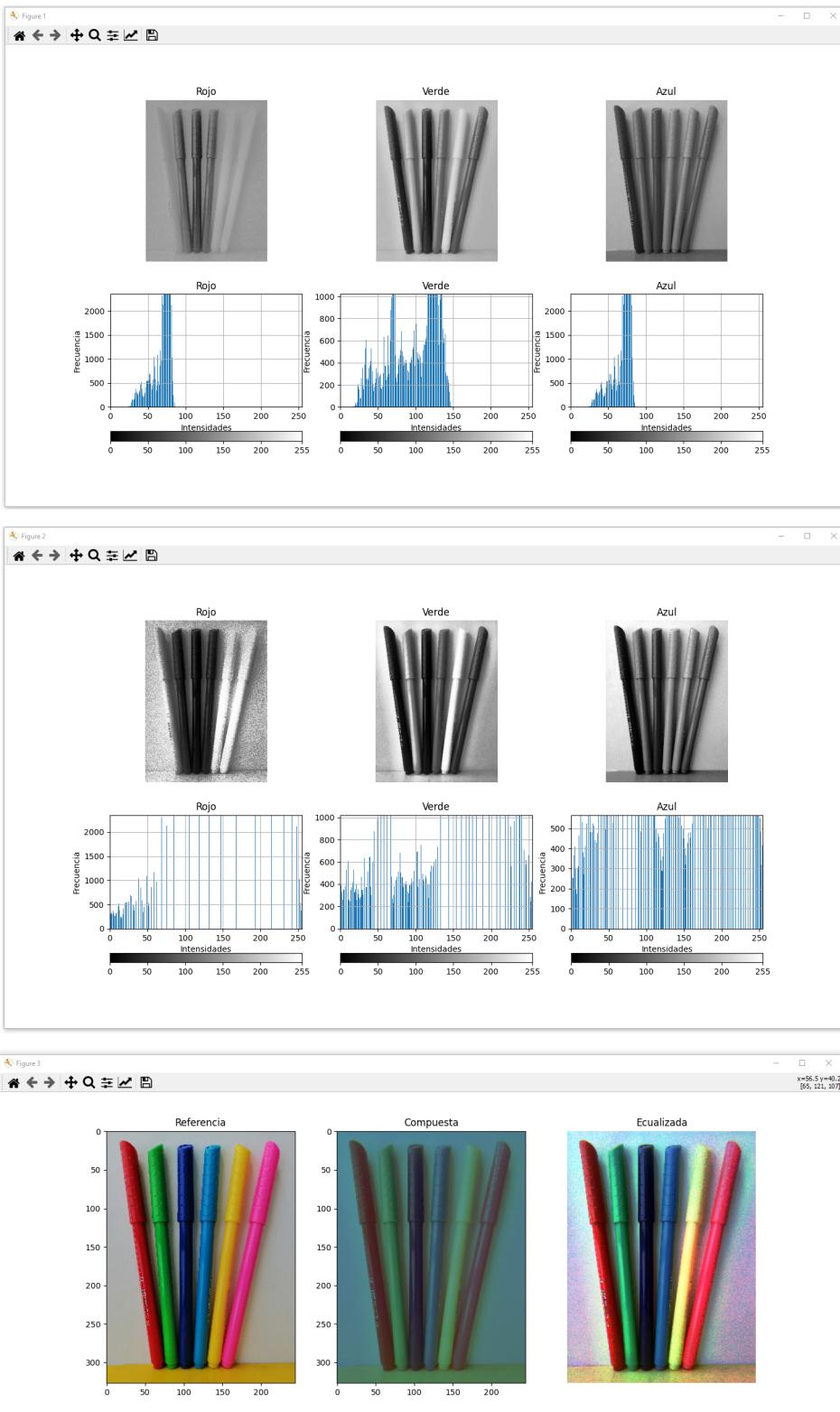


Figura 4.27: La parte superior ilustra los histogramas de las fotografías tomadas en escala de grises, el la central se aprecia una vez se han ecualizado las intensidades y, en la parte inferior de izquierda a derecha, la imagen de referencia, la compuesta sin ajustar y, finalmente, la ecualizada.

4.4. Histograma por Especificación

El ajuste del histograma por especificación es una técnica que se utiliza para acercar la forma del histograma de una imagen de entrada a otro de referencia. Su objetivo es cambiar la distribución de frecuencia de los niveles de gris de una imagen para mejorar su calidad visual o para ajustarla cuando se desean unir varias fotografías y se requiere similitudes en brillo y contraste. En general, se puede aplicar para emparejar características de dos o más imágenes, especialmente cuando estas provienen de distintas fuentes o se han capturado en diversas condiciones, como variaciones en la luz. Cada canal de color de las imágenes se empareja de manera individual. La transformación se realiza usando una función de transformación que mapea los valores de los niveles de gris de la imagen de entrada en los valores deseados del histograma de referencia. La coincidencia de histogramas por especificación busca encontrar una transformación que permita que el histograma de una imagen discreta de entrada corresponda lo más cercanamente posible con el histograma especificado de referencia [5, 13, 33].

4.4.1. Algoritmo Histograma por Especificación usando el Mayor Vecino más Cercano

A partir del histograma de una imagen de referencia R , se buscar modificar la imagen de entrada E generando una nueva imagen de salida S cuya salida, en términos del histograma, es similar a R . Los pasos del algoritmo son [5, 8, 13]:

1. Calcule el histograma acumulado de probabilidad $Hr(n)$ de la imagen de referencia R mediante el siguiente proceso: se cuentan las frecuencias de los niveles de intensidad presentes en la imagen de referencia R . Posteriormente se evalúan las probabilidades de cada nivel de intensidad dividiendo la frecuencia de cada nivel por el número total de píxeles en la imagen de referencia R . Seguidamente, se hagan las probabilidades acumuladas $Hr(n)$ sumando las probabilidades de todos los niveles de intensidad menores o iguales al nivel de intensidad actual n .
2. Realice, de manera similar, el proceso para la imagen de entrada E hasta obtener el histograma de probabilidad acumulado $He(k)$, pero con sus propios datos.
3. Es necesario establecer una correspondencia entre los niveles de intensidad de la imagen de entrada E y la de referencia R , para obtener la tabla de búsqueda y reemplazo. Para lograrlo, se realizan los siguientes pasos:
 - a) Para cada intensidad k en el histograma de probabilidad acumulado de entrada $He(k)$, se busca la intensidad n en el histograma de referencia $Hr(n)$ cuyo valor acumulado sea igual o superior al valor acumulado en $He(k)$. Si hay varios iguales, se toma el primero que aparece en la lista de menor a mayor en la búsqueda. Véase la Figura 4.28.
 - b) Repita los pasos para todos los valores de k en $He(k)$, asegurándose de que todas las intensidades k en la imagen de entrada E tengan una intensidad n asociada en la imagen de referencia R . Esta relación crea una tabla de búsqueda y equivalencia entre los niveles de las intensidades de ambas imágenes.
4. Crea una copia de la imagen de entrada E en una nueva denominada S .
5. Para cada intensidad de píxel en la nueva imagen S , sustituya el valor de intensidad k del píxel con el valor de intensidad n encontrado en la tabla de búsqueda y equivalencia. La imagen transformada S tiene el histograma $Hs(n)$ similar al deseado de referencia $Hr(n)$.

Al aplicar el ajuste por especificación, la imagen de referencia R no se requiere tener la misma resolución que la imagen de entrada E a ajustar. Esto significa que las dimensiones en píxeles (ancho y alto) de ambas imágenes pueden ser diferentes. Sin embargo, un requisito fundamental es que tanto la imagen de entrada como la imagen de referencia comparten la misma profundidad de color, que se refiere al número de bits utilizados para codificar la información de color de cada píxel en la imagen. Por ejemplo, las imágenes de 8 bits permiten un total de $2^8 = 256$ tonalidades distintas por componente del color. Esto se necesita para garantizar una correspondencia adecuada en el rango de valores de intensidad y, en consecuencia, una especificación adecuada de las características de tonalidad y color deseadas. Si las profundidades de color difieren, los valores de intensidad no serían comparables, lo que lleva a resultados indeseados.

4.4.2. Algoritmo Histograma por Especificación usando Interpolación Lineal

En el método explicado se utilizó la estrategia del mayor vecino más cercano para establecer la correspondencia entre los niveles de intensidad de la imagen de entrada R y la imagen de referencia E . Si bien esta aproximación es eficiente desde el punto de vista computacional, una alternativa más adecuada, aunque más costosa computacionalmente, sería utilizar una interpolación. Para lograrlo, en lugar de simplemente asignar el nivel de intensidad n mayor y más cercano del histograma acumulado de referencia $Hr(n)$, la interpolación permite calcular un valor de intensidad intermedio que se aproxime de manera más suave a la distribución de intensidades deseada. Por ejemplo, en lugar de asignar directamente el nivel de intensidad n de la imagen del histograma acumulado referencia $Hr(n)$ al nivel k del histograma acumulado de entrada $He(k)$, se puede realizar una interpolación lineal entre los dos niveles de intensidad más cercanos n_1 y n_2 en el histograma de referencia, utilizando el valor acumulado de probabilidad de k como factor de ponderación.

La interpolación lineal se emplea para determinar el nivel de intensidad interpolado n entre dos niveles consecutivos n_1 y n_2 en el histograma de referencia, basándose en el valor acumulado de probabilidad correspondiente $He(k)$ de la imagen de entrada. El proceso se puede describir de la siguiente manera:

1. Se calculan los histogramas acumulados $He(k)$ para la imagen de entrada y $Hr(n)$ para la imagen de referencia, donde cada uno representa la suma acumulativa de las probabilidades hasta cada nivel de intensidad.
2. Para cada nivel de intensidad k en la imagen de entrada E , se identifican n_1 y n_2 en el histograma de referencia $Hr(n)$ de tal forma que n_1 sea el mayor nivel cuya probabilidad acumulada es menor o igual a $He(k)$, y n_2 es el menor nivel cuya probabilidad acumulada es mayor o igual a $He(k)$.
3. Se utiliza la expresión 4.30 de interpolación lineal para calcular el valor de n :

$$n = \left[\frac{he(k) - H_r(n_1)}{H_r(n_2) - H_r(n_1)} \right] (n_2 - n_1) + n_1 \quad (4.30)$$

donde:

- $He(k) - Hr(n_1)$ es la diferencia entre el valor acumulado de k y la probabilidad acumulada en n_1 .
 - $Hr(n_2) - Hr(n_1)$ es la diferencia total de probabilidades acumuladas entre n_1 y n_2 .
 - El término fraccional escala la diferencia $n_2 - n_1$ basándose en la posición relativa de $He(k)$ entre $Hr(n_1)$ y $Hr(n_2)$.
 - La suma de n_1 ajusta el resultado al rango correcto de intensidades.
4. Crea una copia de la imagen de entrada E en una nueva denominada S .
 5. Para cada intensidad de píxel en la nueva imagen S , sustituya el valor de intensidad k del píxel con el valor de intensidad n encontrado en la tabla de búsqueda y equivalencia al evaluar en (4.30) para obtener el n . La imagen transformada S tiene el histograma $Hs(n)$ similar al deseado de referencia $Hr(n)$.

Este método proporciona una transición suave entre los niveles de intensidad, evitando discontinuidades abruptas en la distribución de intensidades de la imagen transformada y garantizando una mayor fidelidad visual. Aunque este enfoque de interpolación requiere cálculos adicionales para cada nivel de intensidad k , puede producir resultados más exactos y suaves en la transformación del histograma, evitando discontinuidades bruscas en la distribución de la imagen de salida S .

Es importante tener en cuenta que la elección entre el método del vecino más cercano y la interpolación implica un equilibrio entre la eficiencia computacional y la calidad de la transformación del histograma. En aplicaciones donde se requiere una mayor exactitud, la interpolación puede ser la opción preferida, mientras que cuando el rendimiento es crítico, el método del mayor vecino puede ser una alternativa más adecuada.

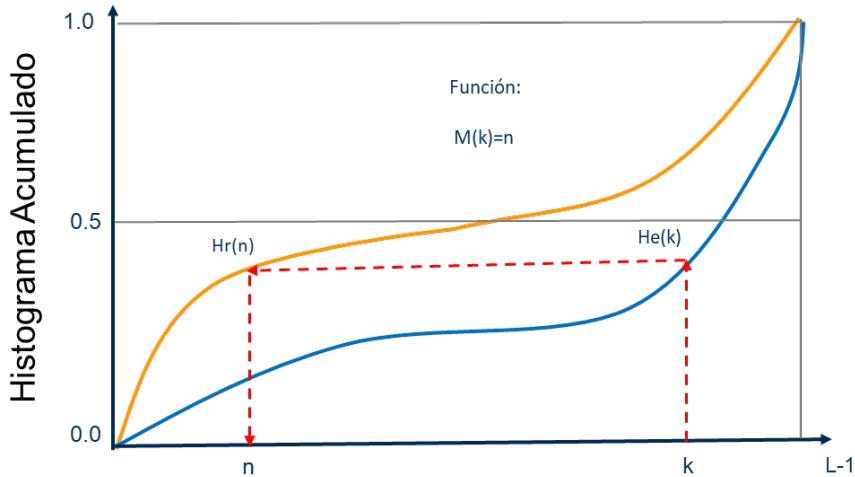


Figura 4.28: Transformación del histograma por especificación.

4.4.3. Ejemplo en Clase: Ajuste por Especificación

Se dispone de dos imágenes: una de referencia R y una imagen de entrada E . El objetivo es modificar la imagen de entrada E para que su apariencia se asemeje a la imagen de referencia R , particularmente en términos de distribución de intensidades y contraste. Para lograr esto, se usa el algoritmo por especificación de histograma, el cual ajusta la distribución de intensidades de la imagen de entrada E para que coincida con la distribución de intensidades de la imagen de referencia R .

$$\text{Imagen de entrada } E: \begin{bmatrix} 1 & 4 & 3 & 5 \\ 7 & 7 & 2 & 7 \\ 4 & 7 & 4 & 5 \\ 5 & 7 & 3 & 4 \end{bmatrix} \quad \text{Imagen de referencia } R: \begin{bmatrix} 2 & 5 & 6 & 6 \\ 4 & 5 & 6 & 3 \\ 4 & 5 & 3 & 4 \\ 2 & 3 & 5 & 4 \end{bmatrix}$$

Figura 4.29: Imágenes de entrada E y referencia R de 3 bits y 4×4 para el ajuste por especificación.

Ahora, dadas las dos matrices de las imágenes de entrada E y referencia R , como se representan en Figura 4.29, se procede a obtener las Tablas 4.5 y 4.4, que corresponden a los pasos 1 y 2 del algoritmo, para las probabilidades acumuladas $Hr(n)$ y $He(k)$.

Clase n	Frecuencia h_r	Probabilidad p_r	Probabilidad acumulada $Hr(n)$
0	0	0.000	0.0000
1	0	0.000	0.0000
2	2	0.1250	0.1250
3	3	0.1875	0.3125
4	4	0.2500	0.5625
5	4	0.2500	0.8125
6	3	0.1875	1.0000
7	0	0.0000	1.0000

Cuadro 4.4: Tabla de frecuencias y probabilidades para la imagen referencia R .

Clase k	Frecuencia h_e	Probabilidad p_e	Probabilidad acumulada $He(k)$
0	0	0.0000	0.0000
1	1	0.0625	0.0625
2	1	0.0625	0.1250
3	2	0.1250	0.2500
4	4	0.2500	0.5000
5	3	0.1875	0.6875
6	0	0.0000	0.6875
7	5	0.3125	1.0000

Cuadro 4.5: Tabla de frecuencias y probabilidades para la imagen de entrada E .

Antes de realizar el proceso de especificación, se procede a graficar los histogramas de las imágenes de referencia R y de entrada E . De esta manera, se puede contrastar visualmente la distribución de las intensidades en las imágenes antes y después de aplicar el algoritmo. Esta comparación permite evaluar la efectividad del proceso y apreciar las mejoras en contraste y apariencia visual en la imagen de salida S ajustada. Véase la Figura 4.30.

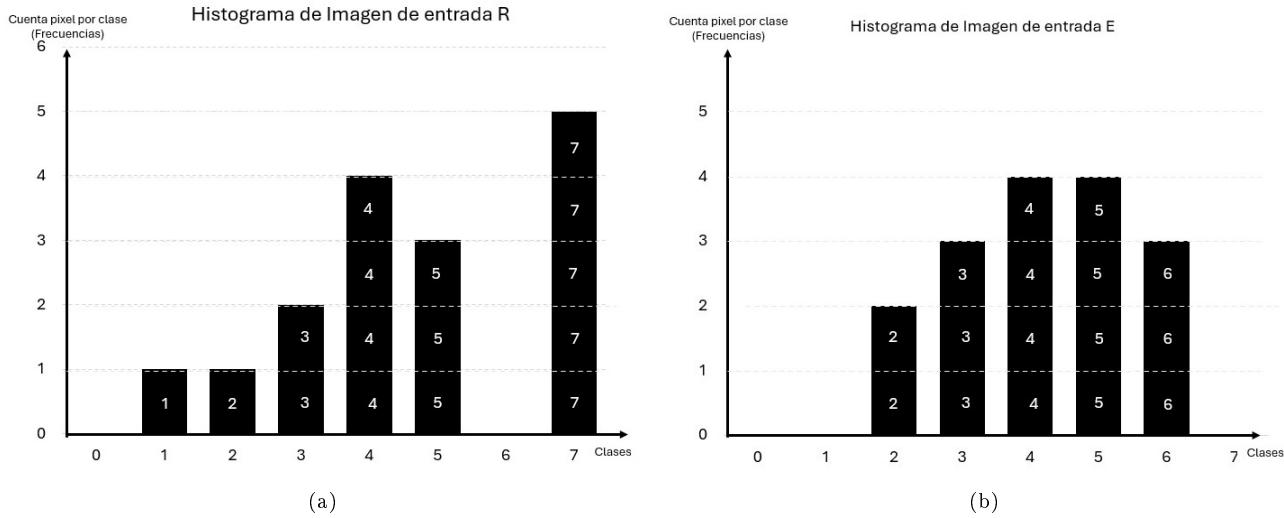


Figura 4.30: a) Histograma Imagen de referencia R vs b) Histograma de Imagen de entrada E.

Con las probabilidades acumuladas se procede a realizar el paso 3 del algoritmo. De manera ilustrativa, a partir de la Tabla 4.5, la primera probabilidad acumulada significativa corresponde a la clase $k = 1$ de la imagen de entrada E ; es decir $He(1) = 0.0625$. Al examinar la Tabla 4.4 de la imagen de referencia R , se encuentra que la primera clase n con una probabilidad acumulada mayor o igual es $Hr(2) = 0.1250$ la cual corresponde a la clase $n = 2$. Este proceso se repite para todos los valores en la tabla $He(k)$, hasta realizar la especificación de histograma hasta obtener la Tabla 4.6 de equivalencias de las intensidades en entrada k con respecto a las de referencia n .

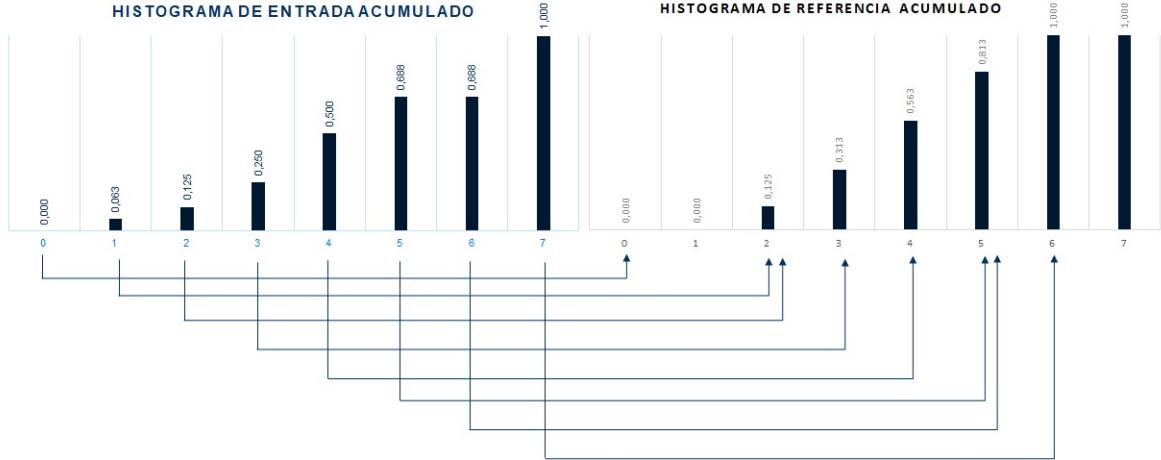


Figura 4.31: Correspondencia entre histogramas acumulados de la imagen de entrada E y la imagen de referencia R .

Clase k	Nueva Clase n
0	0
1	2
2	2
3	3
4	4
5	5
6	5
7	6

Cuadro 4.6: Tabla búsqueda con las nuevas intensidades equivalentes. Las clases k de la imagen de entrada E , se reemplazan por los valores adyacentes de la nueva clase n .

De manera alternativa, la gráfica mostrada en la Figura 4.31 ilustra las correspondencias entre las intensidades de las clases de entrada k y su salida n . Cada columna está relacionada a través de una flecha que aproxima los histogramas acumulados.

Como se puede observar en Figura 4.32, el nuevo histograma de salida presenta una forma similar al de referencia, teniendo una mayor concentración en el centro que en su forma original.

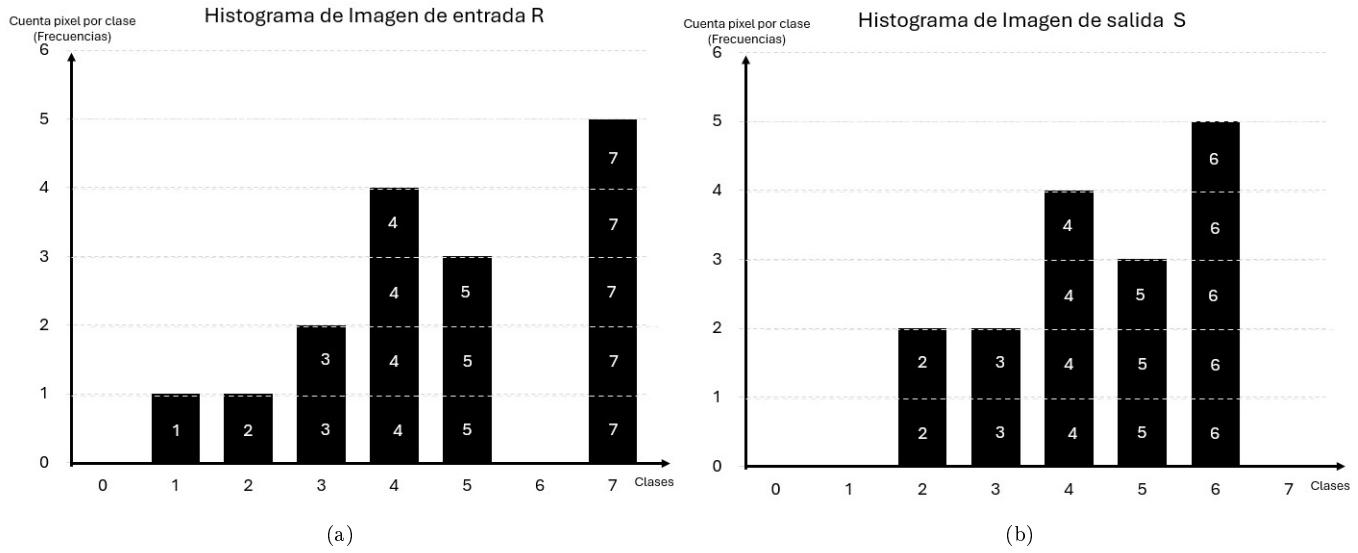


Figura 4.32: a) Histograma Imagen de referencia *R* vs b) Histograma de Salida *S*.

Finalmente, en la Tabla 4.7 se muestra la imagen ajustada por especificación, después de reemplazar los valores de *E* por los indicando en la Tabla 4.6.

$$\begin{bmatrix} 2 & 4 & 3 & 5 \\ 6 & 6 & 2 & 6 \\ 4 & 6 & 4 & 5 \\ 5 & 6 & 3 & 4 \end{bmatrix}$$

Cuadro 4.7: Imagen de salida ajustada por especificación del histograma.

Un histograma es una herramienta útil para entender y mejorar la calidad de una imagen. En la práctica, a menudo queremos modificar el histograma de una imagen para mejorar su calidad visual. La especificación del histograma es una técnica común para lograr esto.

Considérese que se tiene la imagen de Cartagena como referencia *R* y la imagen alterada de Cartagena como entrada *E* tal como se aprecia en la Figura 4.33 Al aplicar la especificación del histograma a la imagen de entrada, se puede ajustar su histograma para que se parezca más al de la imagen de referencia. Es de destacar que en la práctica, la imagen de referencia *R* y la imagen de entrada *E* no tienen que ser las mismas fotografías, ni tampoco de la misma resolución. Solo deben cumplir que sean de la misma cantidad de bits para que sus histogramas sean congruentes.

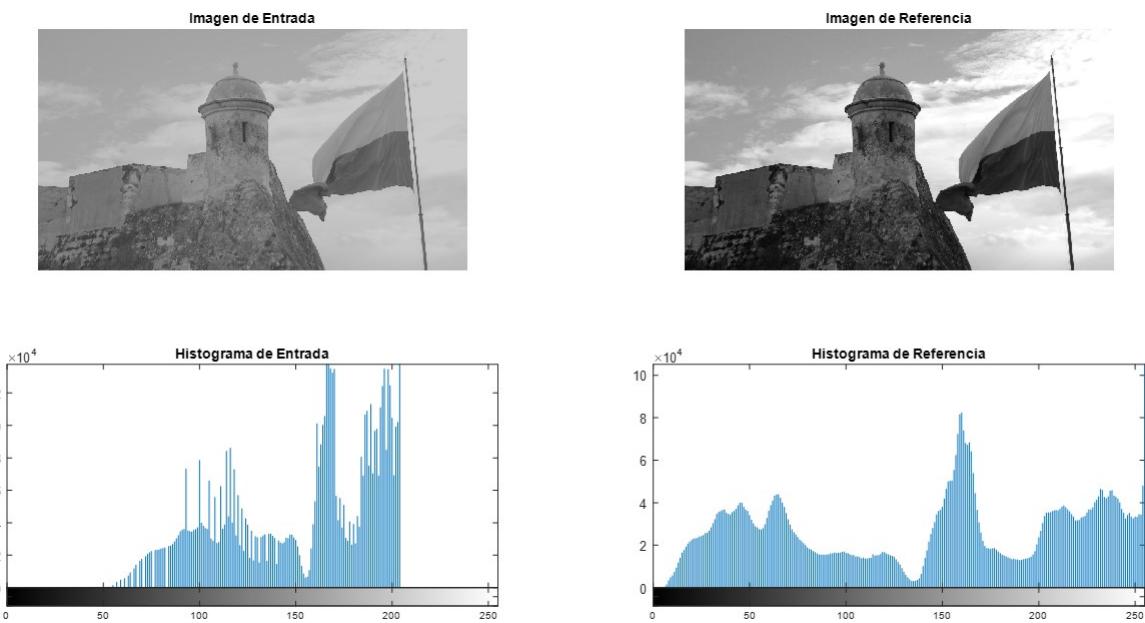


Figura 4.33: Histograma de Referencia vs Imagen de Entrada.

El proceso de especificación del histograma implica asignar nuevos valores de intensidad a los píxeles de la imagen, de modo que el histograma resultante de salida se ajuste a un histograma de referencia. De esta manera, se puede aumentar el contraste y mejorar la apariencia visual de la imagen. Véase la Figura 4.34.

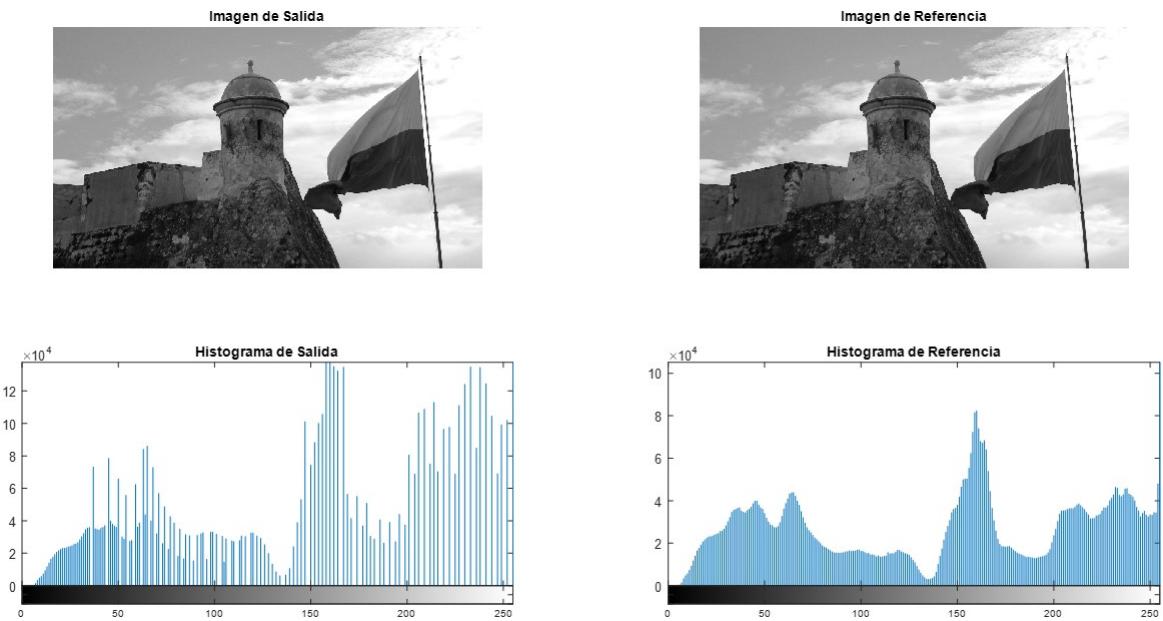


Figura 4.34: Histograma de Salida vs Histograma de Entrada.

4.4.4. Ejemplo en Python: Ecualización por especificación

Este script en Python realiza un ajuste de histograma por especificación entre dos imágenes, permitiendo que la distribución de niveles de intensidad de una imagen de entrada se asemeje al histograma de una imagen de referencia. Este enfoque es útil para ajustar imágenes tomadas en condiciones diferentes, facilitando su combinación o comparación. El proceso comienza cargando las imágenes y seleccionando la misma capa de ambas imágenes. A

continuación, se realiza un recorte (slicing) en la imagen de entrada para reducir la cantidad de datos procesados, optimizando el rendimiento para efectos del ejemplo.

Posteriormente, se implementa la técnica de emparejamiento de histogramas para ajustar la imagen de entrada al histograma de la imagen de referencia.

Solución

```
1 #Importación de Librería Básicas
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import sys
5 from ip_functions import *
6
7 """#Cargar las Imágenes"""
8 # Abrir Imagen
9 RGBr=plt.imread("negativo_chica.jpg")
10 RGBr=255-np.array(RGBr);
11 #Seleccionar la capa Roja Positiva
12 ref=RGBr[:, :, 0]
13 RGBe=plt.imread("cartagena.jpg")
14 #Seleccionar la capa Roja Positiva
15 rent=RGBe[:, :, 0]
16
17
18 #Realizar Slicing para reducir procesamiento
19 S=15
20 rent=np.array(rent[1::S, 1::S])
21
22 plt.figure(figsize=(15,15))
23 plt.subplot(2,2,1)
24 plt.imshow(ref, cmap='gray')
25 plt.axis('off')
26 plt.title('Referencia')
27
28 plt.subplot(2,2,2)
29 plt.imshow(rent, cmap='gray')
30 plt.axis('off')
31 plt.title('Entrada')
32
33 ax3 = plt.subplot(2,2,3)
34 imhist(ref, ax=ax3)
35
36 ax4 = plt.subplot(2,2,4)
37 imhist(rent, ax=ax4)
38 plt.tight_layout()
39 plt.show()
40
41 href=imhist(ref, None, False)
42 Is=histeq(rent, href)
43
44 plt.figure(figsize=(15,15))
45 plt.subplot(2,2,1)
46 plt.imshow(ref, cmap='gray')
47 plt.axis('off')
48 plt.title('Referencia')
49
50 plt.subplot(2,2,2)
51 plt.imshow(Is, cmap='gray')
52 plt.axis('off')
53 plt.title('Salida')
54
55 ax3 = plt.subplot(2,2,3)
56 imhist(ref, ax=ax3)
57
58 ax4 = plt.subplot(2,2,4)
59 imhist(Is, ax=ax4)
60 plt.tight_layout()
61 plt.show()
```

La Fig. 4.35 muestra el proceso de ecualización por especificación de una imagen usando Python. Finalmente, el script genera una visualización. Las funciones referidas allí, se encuentran en el capítulo 13

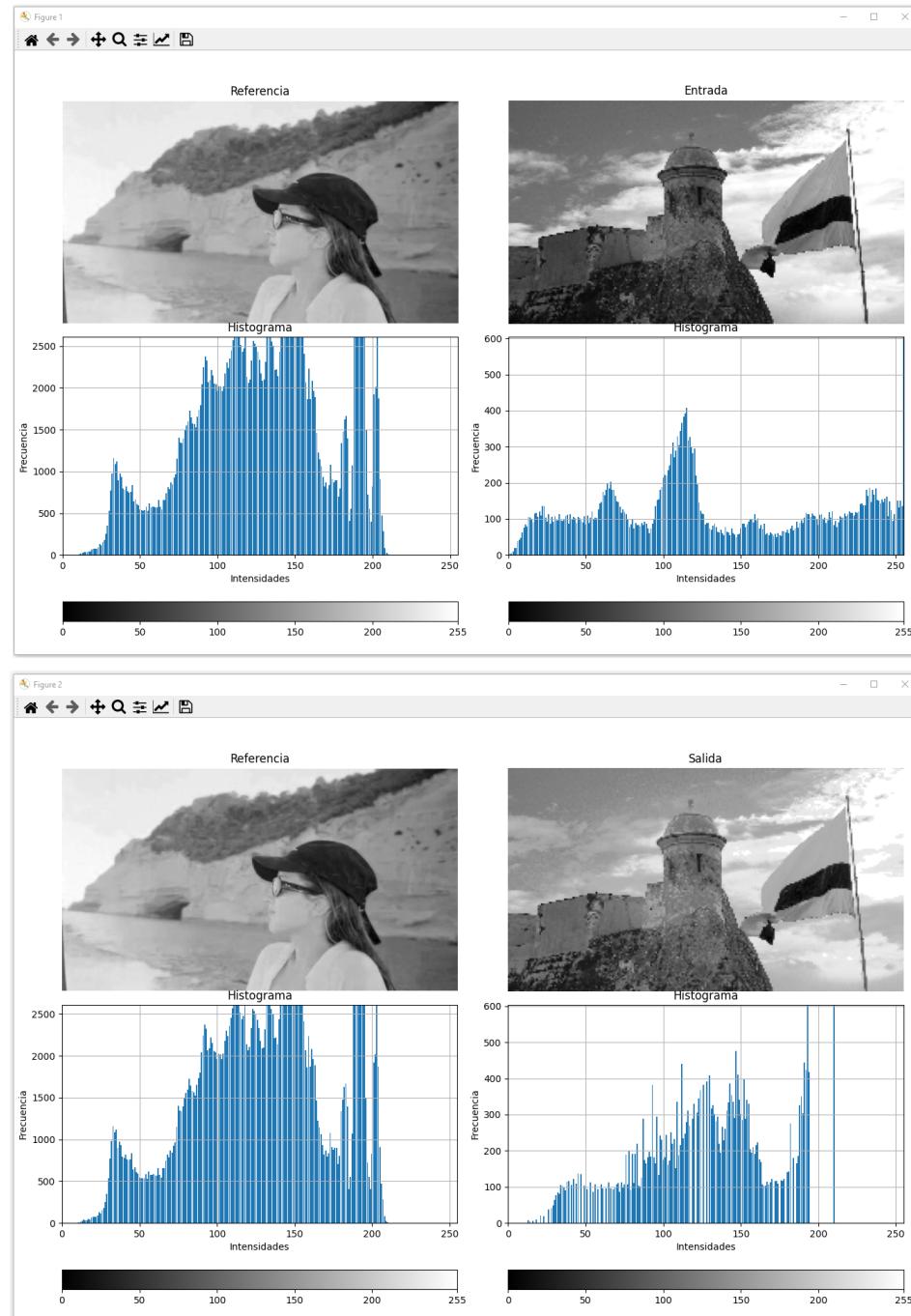


Figura 4.35: Histograma por especificación entre dos imágenes, permitiendo que la distribución de niveles de intensidad de una imagen de entrada se asemeje al histograma de una imagen de referencia.

4.5. Funciones

Las siguientes funciones tienen relación con el tema abordado en este capítulo:

1. `imhist(r, ax=None, ver=True)`: Calcula y muestra el histograma de una imagen en escala de grises `r`. Si `ver=True`, se grafica directamente; si `ver=False`, se devuelve el histograma como arreglo. El parámetro `ax` permite indicar el eje de Matplotlib donde se mostrará la gráfica.
2. `histeq(I, h=None)`: Realiza la **ecualización del histograma** de la imagen `I`, mejorando el contraste mediante la redistribución uniforme de los niveles de intensidad. Si se proporciona el parámetro `h`, ejecuta la **especificación del histograma**, adaptando la imagen de entrada a una distribución de niveles definida externamente.
3. `imadjust(I, in_range=(min, max), out_range=(a, b), gamma=1.0)`: Ajusta los valores de intensidad de los píxeles de la imagen `I` para modificar el contraste. El parámetro `in_range` define el rango de entrada que se desea transformar, `out_range` especifica el nuevo rango de salida, y `gamma` controla la corrección `gamma` para aplicar transformaciones no lineales (por defecto, lineal si `gamma=1.0`).
4. `stretchlim(I, tol=0.01)`: Calcula los límites inferior y superior de intensidad de la imagen `I` para estirar su histograma y mejorar el contraste. El parámetro `tol` (opcional) indica el porcentaje de píxeles a descartar en los extremos del histograma. Si no se proporciona, se asume un valor por defecto de 0.01.

4.6. Preguntas

4.6.1. Preguntas sobre hechos indicados en el texto:

1. ¿Qué representa cada entrada en un histograma de una imagen en escala de grises?
2. ¿Qué parámetros se requieren para realizar el ajuste de histograma mediante una función potencia?
3. ¿Qué es el gamma en el contexto del procesamiento de imágenes?
4. ¿Qué significa que un histograma esté normalizado?
5. ¿Qué valores se deben determinar en el histograma antes de realizar su ecualización?

4.6.2. Preguntas de análisis y comprensión con base en el texto:

1. ¿Cómo se puede utilizar el histograma para estimar el brillo y contraste de una imagen?
2. Analiza las diferencias entre el ajuste de histograma mediante una función exponencial y una logarítmica.
3. ¿En qué consiste la ecualización de histograma?
4. ¿Cómo permite mejorar el contraste de una imagen?
5. Explica en qué consiste la especificación de histograma y cuándo puede ser útil aplicarla.

4.6.3. Ejercicios numéricos

1. Para la matriz de la imagen de 4×4 pixels de 4 bits mostrada en [4.31](#), realice de forma manuscrita:

$$\begin{bmatrix} 5 & 7 & 13 & 9 \\ 12 & 11 & 14 & 10 \\ 13 & 9 & 4 & 12 \\ 9 & 9 & 12 & 7 \end{bmatrix} \quad (4.31)$$

- El histograma de frecuencias y probabilidad.
 - Calcule el brillo y el contraste por los pixeles.
 - Calcule el brillo y el contraste a partir de su histograma.
2. Para la matriz de la imagen de 4×4 pixels de 4 bits mostrada en 4.31, realice el ajuste del histograma de forma lineal $n = 1$.
3. Para la matriz de la imagen de 4×4 pixels de 4 bits mostrada en 4.31, realice de forma manuscrita la ecualización.
4. De manera analítica encuentre la ecuación que permite realizar el ajuste exponencial:

$$I_S = \frac{(S_M - S_m)}{(e^{(E_M - E_m)} - 1)} [e^{(I_E - E_m)} - 1] + S_m$$

5. De manera analítica encuentre la ecuación que permite realizar el ajuste logarítmico:

$$I_S = \frac{(S_m - S_m)}{\ln(E_M - E_m + 1)} \ln[I_E - E_m + 1] + S_m$$

6. A partir de las imágenes de 4×4 y de 4 bits propuestas, realizar la ecualización por especificación del histograma de la imagen de entrada E propuesta en 4.32 con la referencia R ilustrada en 4.33.

$$E = \begin{bmatrix} 11 & 13 & 0 & 13 \\ 7 & 0 & 2 & 5 \\ 15 & 15 & 1 & 15 \\ 15 & 4 & 11 & 0 \end{bmatrix} \quad (4.32)$$

$$R = \begin{bmatrix} 13 & 14 & 2 & 14 \\ 10 & 2 & 5 & 9 \\ 15 & 15 & 3 & 15 \\ 15 & 8 & 13 & 1 \end{bmatrix} \quad (4.33)$$

Nota: Para todos los casos haga el histograma y la matriz de imagen de salida.

Capítulo 5

Modelos básicos del Color

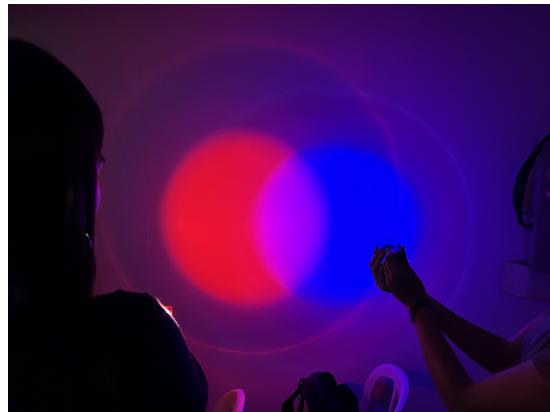
5.1. El concepto del color

La visión del color es una capacidad perceptiva que surge en los animales cuyos sistemas nerviosos están organizados para codificar y evaluar variaciones en las distribuciones espectrales de la luz que llega a los ojos. El resultado de este proceso es una rica fuente de información que se usa para la toma de decisiones.

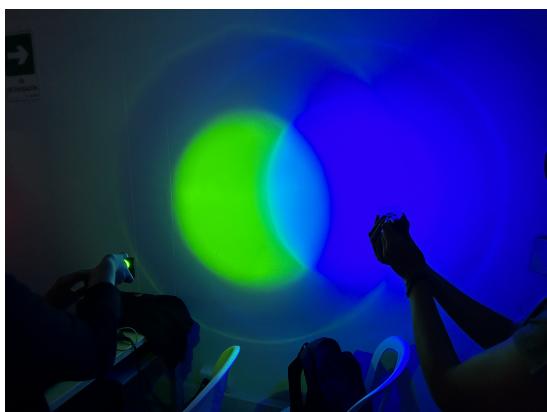
Es difícil señalar puntualmente una razón particular del origen de la visión del color en los animales. Sin embargo, una de las explicaciones dadas indica que podría haber aparecido como una adaptación específica para observar las plantas y los animales. Una teoría simplificada de la visión del color expresa que hay tres colores primarios correspondientes a las clases de conos en la retina de ojo. Los miles de tonos entre los que se puede distinguir se crean mediante la combinación de estímulos de los conos especializados para percibir longitudes de onda del color rojo, el verde y el azul. Por ejemplo, cuando el rojo y el verde se superponen de forma aditiva, con la misma proporción de intensidades, se aprecia el amarillo, como se ve en la Figura 5.1a. Si se combinan independientemente de las fuentes roja y azul, véase la Figura 5.1b, se produce una tonalidad magenta, y el verde y el azul causan el cian como se ven en la Figura 5.1c. El blanco, como se ilustra en la Figura 5.1d, puede generarse cuando las tres fuentes están superpuestas con la misma intensidad. Es de destacar que no existe un conjunto único de tres colores primarios, no obstante, el modelo de rojo, verde y azul (RGB) es el más aceptado por la naturaleza del ojo humano. Parecería que todos los tonos se pueden producir agregando tres fuentes de colores primarios en diversas proporciones, pero hay indicios de que la visión del color es más compleja [20].



(a) La combinación aditiva de rojo y verde produce amarillo.



(b) La combinación aditiva de rojo y azul produce cian.



(c) La combinación aditiva de rojo y verde produce amarillo.



(d) La combinación aditiva de rojo, azul y verde produce blanco.

Figura 5.1: Combinación aditiva del modelo básico rojo, verde y azul como fuente para generar más colores.

5.2. Modelos del Color

5.2.1. Modelo de color CIE XYZ

La CIE (Comision Internationale de l'Eclairage - Comisión Internacional de la Iluminación) fue fundada en 1913 con el objetivo de ser una junta internacional autónoma para discutir y establecer normas relacionadas con la iluminación. El modelo de color CIE RGB se desarrolló para ser completamente independiente de cualquier dispositivo u otro medio de emisión o reproducción, basándose lo más cerca posible en cómo los seres humanos perciben el color. Este tiene dos especificaciones para un observador estándar: la especificación original de 1931 y la especificación revisada de 1964. El observador de 1931 tenía un campo de visión de 2 grados, por lo que se consideró posteriormente inadecuado en muchos casos, ya que no tomó suficientemente en cuenta la visión periférica del observador. La especificación de 1964 amplió el campo de visión del observador a 10 grados con el fin de obtener valores tristimulus que reflejaran una mayor sensibilidad de la retina [10, 58].

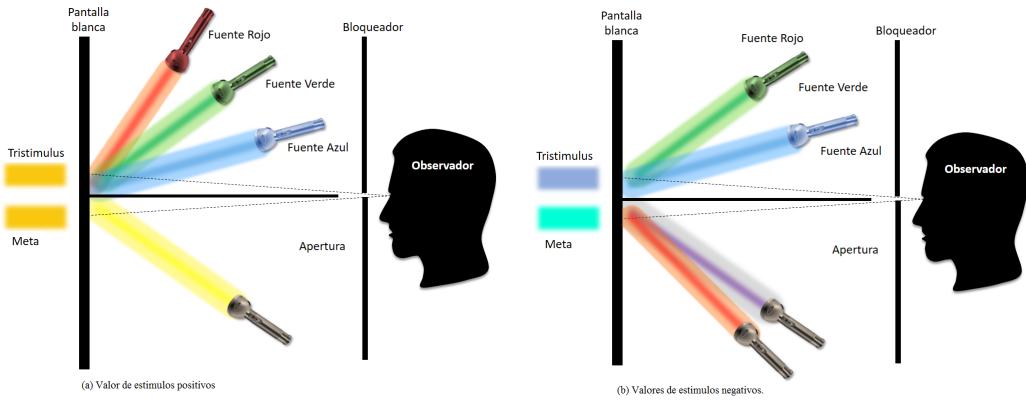


Figura 5.2: Experimento CIE RGB. (a) Valores de estímulos RGB positivos. (b) Valores de estímulos RGB negativos.

En el experimento que se muestra en la Figura 5.2 se mostraban dos colores al observador. Uno de los colores era denominado la meta y el otro la fuente, esta última formada realmente por lámparas de colores base rojo, verde y azul a los cuales se les podía realizar ajustes de intensidad. El experimento consistía el cambiar los valores de las lámpara roja, verde y azul de forma independiente hasta que el observador considerara que había logrado el mismo color propuesto como meta. En ambos casos el observador estándar estaba formado por pequeños grupos de personas (aproximadamente entre 15 y 20) que representaban la visión normal de color humano. Ambas especificaciones utilizaron una técnica similar para combinar los colores con un valor de tristimulus RGB equivalente. La percepción visual de un estímulo neutro (color blanco o gris) corresponde a una respuesta equilibrada de los tres conos diferentes. Hay que notar que para las longitudes de onda inferiores a 550nm, se añaden valores “negativos”. Esto realmente no significaba agregar valores negativos de luz de cierta longitud de onda, se refería al hecho de adicionar algo del color fuente base al color meta para lograr la igualdad. Se debía a que es imposible emparejar un color monocromático por debajo de 550 nm añadiendo tres primarios monocromáticos. Por lo tanto, en la configuración mostrada en la Figura 5.3, se obtendría la coincidencia de percepciones de color en las dos mitades del campo de observación para algunos colores si se añade luz de una de las primarias a la "luz de prueba" [9].

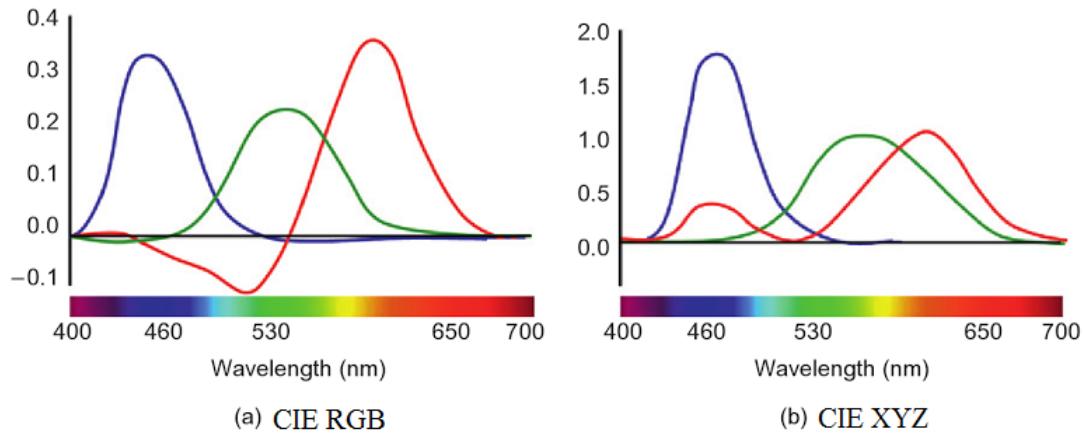
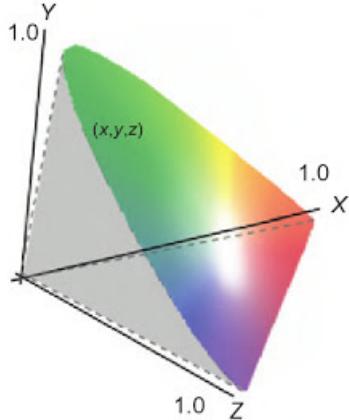


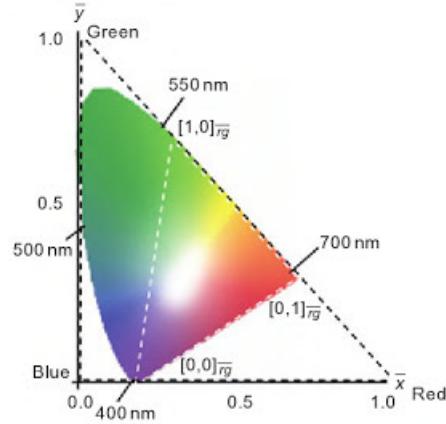
Figura 5.3: Curvas de los modelos (a) CIE RGB y (b) CIE XYZ.

Una de las conclusiones más importantes del experimento es el hecho que no todos los colores pueden ser representados adecuadamente usando las tres fuentes de luz aditivas definidas como rojo, verde y azul. Los resultados generales del experimento se conocen como los Datos de Wright-Guild.

Por lo anterior, fue necesario implementar el modelo teórico CIE XYZ con una descripción que removiera la limitación de los valores negativos de la longitud de onda para representar colores por debajo de los 550nm.



(a) Modelo del color XYZ



(b) Diagrama cromático XYZ

Figura 5.4: (a) Modelo del color XYZ y (b) Diagrama cromático XYZ.

5.2.2. Modelo de color RGB

El esquema de color RGB codifica colores como combinaciones de los tres colores primarios: rojo (R), verde (G), y azul (B). Este es ampliamente utilizado para la transmisión, la representación y el almacenamiento de imágenes a color tanto en dispositivos analógicos, como aparatos de televisión y dispositivos digitales, como computadoras, cámaras digitales y escáneres entre otros. Por esta razón, muchos de los programas de procesamiento de imágenes y gráficos, y la mayoría de las librerías de lenguaje de programación, utilizan el esquema RGB como su representación interna básica para la manipulación de las imágenes a color [13, 50].

El modelo RGB es un sistema de color aditivo, lo que significa que todos los colores comienzan con negro y se crean mediante la adición de los colores primarios. Se puede pensar en la formación de color en este sistema como algo que ocurre en un cuarto oscuro donde se puede superponer tres haces de luz (uno rojo, uno verde y uno azul) sobre una hoja de papel blanco. Para crearlos diferentes colores, se debe modificar la intensidad de cada uno de estos haces de luz monocromática de forma independiente. La intensidad distinta de cada haz de color primario controlan el tono y el brillo del color resultante. Los colores grises y blancos son creados por la mezcla de los tres haces de colores primarios en la misma intensidad.

Para el diseño de páginas web, los colores se especifican con RGB. Hoy en día, con el predominio de pantallas de 24 bits, permite ver 16,7 millones de colores. En el diseño de páginas web, hay 216 colores llamados "web-safe" RGB representados por valores hexadecimales, aunque es claro que RGB puede representar muchos más colores.

El espacio de color RGB puede visualizarse como un cubo unitario tridimensional en el cual los tres colores primarios forman el eje de coordenadas. Los valores RGB son positivos y se encuentran en el rango de $[0, C_{max}]$. Para la mayoría de imágenes digitales, $C_{max} = 255$. Cada color posible C_i corresponde a un punto dentro del cubo de color RGB de la forma:

$$C_i = (R_i, G_i, B_i) \quad (5.1)$$

donde $0 \leq R_i, G_i, B_i \leq C_{max}$. Los valores RGB son frecuentemente normalizados al intervalo $[0, 1]$ de manera que el espacio de color resultante forma un cubo unitario. Véase la Figura 5.5.

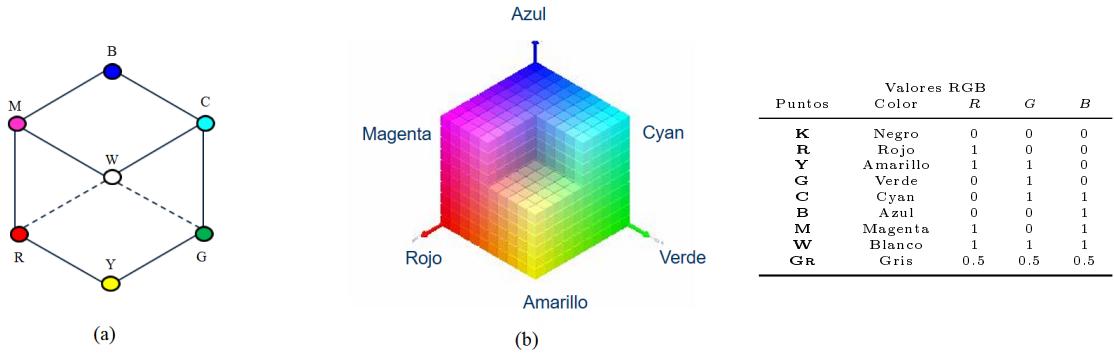


Figura 5.5: Representación del espacio de color RGB como un cubo unitario tridimensional.

El punto $S = (0, 0, 0)$ corresponde al color negro, $W = (1, 1, 1)$ al blanco, y todos los puntos que se encuentran en la diagonal entre S y W son tonos de gris creados a partir de componentes iguales de color $R = G = B$. La Figura 5.6 muestra una imagen de prueba a color y sus correspondientes componentes de color RGB, mostrados como imágenes de intensidad.



Figura 5.6: Imagen a color y sus correspondientes canales RGB.

5.2.3. Modelo del color CMYK

El modelo sustractivo del color CMYK (Cian, Magenta, Amarillo, Negro) se utiliza en la impresión. En CMYK, cada color primario (cian, magenta y amarillo) se representa mediante la combinación de pigmentos que absorben selectivamente las longitudes de onda de luz. Por ejemplo, el cian absorbe el rojo, el magenta absorbe el verde y el amarillo absorbe el azul. Al combinar estos colores en diferentes proporciones, es posible crear una gama de colores adecuada para la impresión sobre blanco. Al imprimir una imagen en color, se superponen pequeñas gotas de tinta

de cian, magenta, amarillo y negro en el papel en diferentes combinaciones y densidades para crear la gama de colores y tonos. El negro se añade en algunos casos al CMY para mejorar la reproducción de tonos oscuros y reducir el consumo de tintas de color que resultan más costosas.

La Figura 5.7 muestra una imagen de prueba a color y sus correspondientes componentes de color CMY mostrados como imágenes de intensidad.

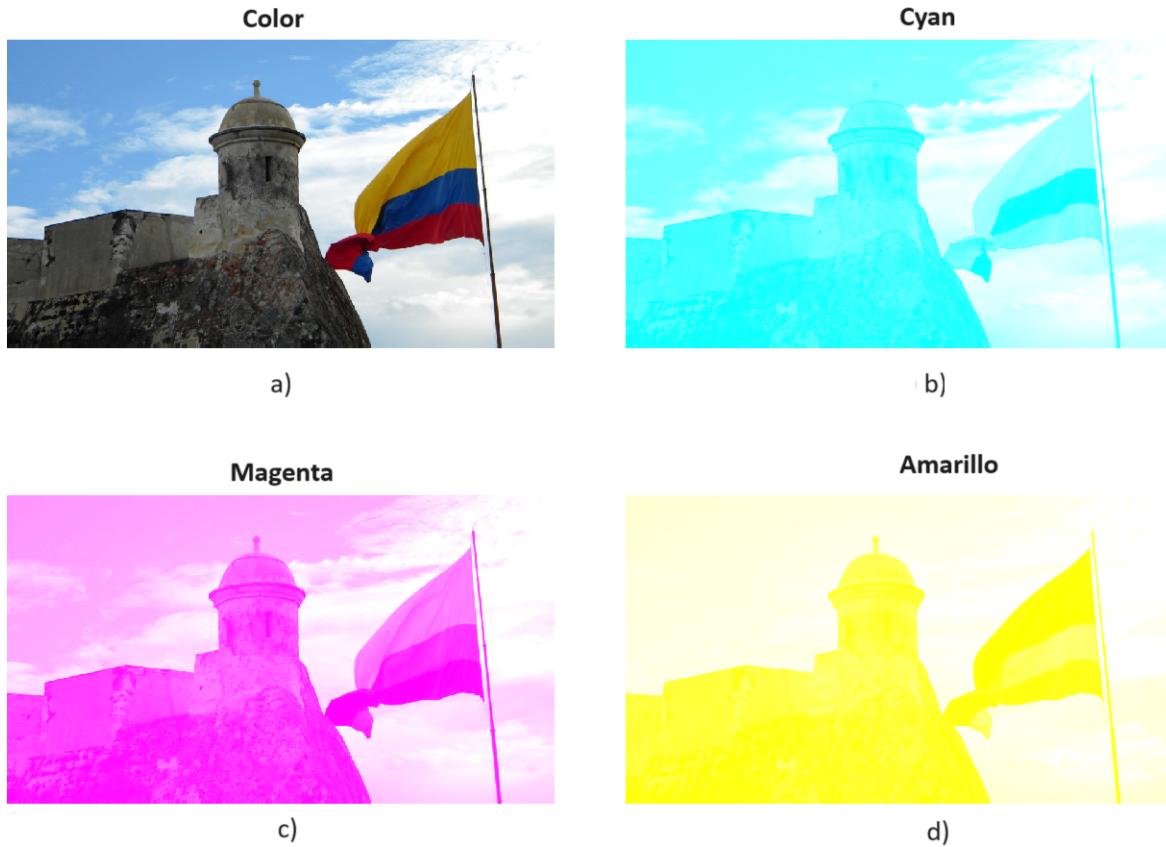


Figura 5.7: Imagen a color y sus correspondientes canales CMY.

5.2.4. Modelo de color HSV

En el espacio de color HSV, los colores son especificados por los componentes *Hue*, *Saturation* y *Value* (*Matiz*, *Saturación* y *Valor*). Con frecuencia, el espacio de color HSV es llamado HSB. Aunque el acrónimo es diferente, *Brightness* (*Brillo*), denota el mismo espacio de color. El espacio de color HSV es tradicionalmente mostrado como una pirámide de seis caras al revés, donde el eje vertical representa el valor de *V* (*Value*), la distancia horizontal del eje el valor de *S* (*Saturation*) y el ángulo el valor de *H* (*Hue*). El punto negro está en la punta de la pirámide y el punto blanco se encuentra en el centro de la base. Los tres colores primarios *Red*, *Green* y *Blue*, y los colores mezclados por parejas *Yellow*, *Cyan* y *Magenta* son los puntos de las esquinas de la base. Mientras este espacio se representa a menudo como una pirámide, de acuerdo con su definición matemática, el espacio es en realidad un cilindro, como se muestra en la Figura 5.8.

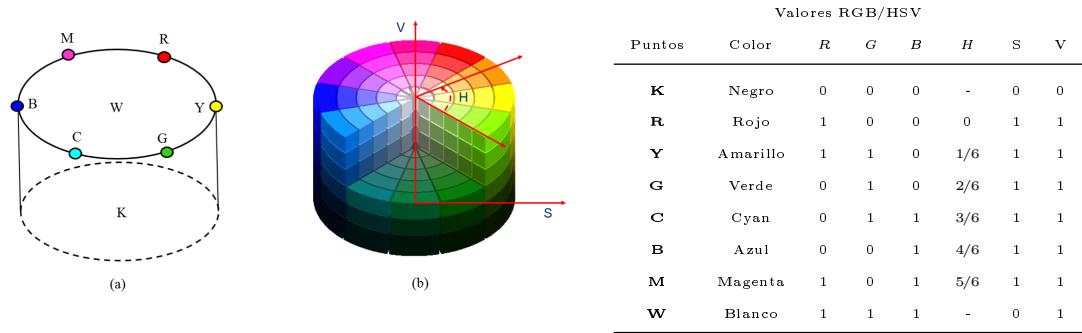


Figura 5.8: Espacio de color HSV.

En la Figura 5.9 se ilustra una representación del modelo HSV para cada una de sus tres capas. La capa *Hue* representa el color, la capa *Saturation* la cantidad del color y *Value* su brillo.



Figura 5.9: Imagen a color y sus correspondientes canales HSV.

5.2.5. Conversión de RGB a CMYK

Los valores de entrada son R , G y B :

$$0 \leq R \leq 255 \quad 0 \leq G \leq 255 \quad 0 \leq B \leq 255$$

Los valores de la imagen deben normalizarse entre 0 y 1, por lo que se dividen entre 255, así:

$$r = \frac{R}{255} \quad g = \frac{G}{255} \quad b = \frac{B}{255}$$

Si no se requiere el negro K , entonces:

$$C = 1 - r \quad M = 1 - g \quad Y = 1 - b$$

Donde los valores de la salida son C , M , y Y :

$$0 \leq C \leq 1 \quad 0 \leq M \leq 1 \quad 0 \leq Y \leq 1$$

Si se necesita el negro K , entonces los valores se calculan como:

$$K = 1 - \max(r, g, b) \quad C = \frac{1 - r - K}{1 - K} \quad M = \frac{1 - g - K}{1 - K} \quad Y = \frac{1 - b - K}{1 - K}$$

Donde los valores de la salida son C , M , Y y K :

$$0 \leq C \leq 1 \quad 0 \leq M \leq 1 \quad 0 \leq Y \leq 1 \quad 0 \leq K \leq 1$$

5.2.6. Conversión de CMYK a RGB

Los valores de entrada son C , M , Y y K :

$$0 \leq C \leq 1 \quad 0 \leq M \leq 1 \quad 0 \leq Y \leq 1 \quad 0 \leq K \leq 1$$

Si no se requiere el negro K , entonces:

$$r = 1 - C \quad g = 1 - M \quad b = 1 - Y$$

Donde los valores de la salida son r , g , y b :

$$0 \leq r \leq 1 \quad 0 \leq g \leq 1 \quad 0 \leq b \leq 1$$

Si se requiere el negro K , entonces los valores se calculan como:

$$R = 255 \cdot (1 - C) \cdot (1 - K) \quad G = 255 \cdot (1 - M) \cdot (1 - K) \quad B = 255 \cdot (1 - Y) \cdot (1 - K)$$

Donde los valores de salida son R , G y B :

$$0 \leq R \leq 255 \quad 0 \leq G \leq 255 \quad 0 \leq B \leq 255$$

5.2.7. Conversión de RGB a HSV

Los valores de entrada son R , G y B :

$$0 \leq R \leq 255 \quad 0 \leq G \leq 255 \quad 0 \leq B \leq 255$$

Los valores de la imagen deben normalizarse entre 0 y 1, por lo que se dividen entre 255, así:

$$r = \frac{R}{255} \quad g = \frac{G}{255} \quad b = \frac{B}{255}$$

Se crean las variables temporales:

$$C_{max} = \max(r, g, b) \quad C_{min} = \min(r, g, b) \quad \delta = C_{max} - C_{min}$$

Para calcular el H :

$$H = \begin{cases} H = 0^\circ & \text{si } \delta = 0 \\ H = 60^\circ \times \text{mod6} \left[\frac{(g-b)}{\delta} \right] & \text{si } C_{max} = r \\ H = 60^\circ \times \left[\frac{(b-r)}{\delta} + 2 \right] & \text{si } C_{max} = g \\ H = 60^\circ \times \left[\frac{(r-g)}{\delta} + 4 \right] & \text{si } C_{max} = b \end{cases}$$

El operador módulo (`mód`) calcula el residuo de la división entre dos números a y b ; es decir, $a \bmod b$ es el residuo cuando a se divide por b . En la conversión del modelo *RGB* al *HSV*, el cálculo de H puede dar como resultado un valor que debería estar en el rango de 0° a 360° , por lo que para asegurarse de esto, se usa la mencionada operación.

Para calcular S :

$$S = \begin{cases} S = 0 & \text{si } C_{max} = 0 \\ S = \frac{\delta}{C_{max}} & \text{si } C_{max} \neq 0 \end{cases}$$

Para calcular V :

$$V = C_{max}$$

Donde los valores de la salida son H , S y V :

$$0^\circ \leq H \leq 360^\circ \quad 0 \leq S \leq 1 \quad 0 \leq V \leq 1$$

Ejemplo de Conversión de RGB a HSV

Para la imagen dada RGB, convertirla a HSV:

$$R = \begin{bmatrix} 255 & 0 \\ 0 & 127 \end{bmatrix} \quad G = \begin{bmatrix} 0 & 255 \\ 0 & 127 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 255 & 127 \end{bmatrix}$$

Solución

Los valores de entrada R , G y B deben estar entre:

$$0 \leq R \leq 255 \quad 0 \leq G \leq 255 \quad 0 \leq B \leq 255$$

Primero, se normalizan los valores R , G y B dividiéndolos por 255:

$$r = \frac{R}{255}, \quad g = \frac{G}{255}, \quad b = \frac{B}{255}$$

Después, se obtiene:

$$r = \begin{bmatrix} \frac{255}{255} & \frac{0}{255} \\ \frac{0}{255} & \frac{127}{255} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0.498 \end{bmatrix} \quad g = \begin{bmatrix} \frac{0}{255} & \frac{255}{255} \\ \frac{255}{255} & \frac{127}{255} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0.498 \end{bmatrix} \quad b = \begin{bmatrix} \frac{0}{255} & \frac{0}{255} \\ \frac{127}{255} & \frac{255}{255} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0.498 \end{bmatrix}$$

Para el cálculo de C_{\max} , C_{\min} y δ de cada pixel se hace:

$$C_{\max} = \max(r, g, b) \quad C_{\min} = \min(r, g, b) \quad \delta = C_{\max} - C_{\min}$$

Para el cálculo de la matriz H , se logra con:

$$H = \begin{cases} H = 0^\circ & \text{si } \delta = 0 \\ H = 60^\circ \times \text{mod6}\left[\frac{(g-b)}{\delta}\right] & \text{si } C_{\max} = r \\ H = 60^\circ \times \left[\frac{(b-r)}{\delta} + 2\right] & \text{si } C_{\max} = g \\ H = 60^\circ \times \left[\frac{(r-g)}{\delta} + 4\right] & \text{si } C_{\max} = b \end{cases}$$

Para el cálculo de la matriz S se realiza:

$$S = \frac{\delta}{C_{\max}}$$

Para el cálculo de la matriz V se obtiene a partir de:

$$V = C_{\max}$$

La Tabla 5.1 resume los cálculos numéricos de C_{\max} , C_{\min} , δ , H , S y V de cada pixel:

Pixel	r	g	b	C_{\max}	C_{\min}	δ	H	H normalizado	S	V
(1, 1)	1	0	0	1	0	1	0°	$\frac{0}{360} = 0.000$	1	1
(1, 2)	0	1	0	1	0	1	120°	$\frac{120}{360} = 0.333$	1	1
(2, 1)	0	0	1	1	0	1	240°	$\frac{240}{360} = 0.666$	1	1
(2, 2)	0.498	0.498	0.498	0.498	0.498	0	0°	$\frac{0}{360} = 0.000$	0	0.498

Cuadro 5.1: Cálculos completos de C_{\max} , C_{\min} , δ , H , S y V .

El resultado en formato HSV es:

$$H = \begin{bmatrix} 0^\circ & 120^\circ \\ 240^\circ & 0^\circ \end{bmatrix} = \begin{bmatrix} 0.00 & 0.33 \\ 0.66 & 0.00 \end{bmatrix} \quad S = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad V = \begin{bmatrix} 1 & 1 \\ 1 & 0.498 \end{bmatrix}$$

5.2.8. Conversión de HSV a RGB

Los valores de entrada son H , S y V deben estar entre:

$$0^\circ \leq H \leq 360^\circ \quad 0 \leq S \leq 1 \quad 0 \leq V \leq 1$$

Se crean las variables temporales C , X , m , r , g y b con:

$$C = S \times V \quad X = C \times \left[1 - \left| \text{mod2}\left(\frac{H}{60^\circ}\right) - 1 \right| \right] \quad m = V - C$$

$$(r, g, b) = \begin{cases} (C, X, 0) & \text{si } 0^\circ \leq H < 60^\circ \\ (X, C, 0) & \text{si } 60^\circ \leq H < 120^\circ \\ (0, C, X) & \text{si } 120^\circ \leq H < 180^\circ \\ (0, X, C) & \text{si } 180^\circ \leq H < 240^\circ \\ (X, 0, C) & \text{si } 240^\circ \leq H < 300^\circ \\ (C, 0, X) & \text{si } 300^\circ \leq H < 360^\circ \end{cases}$$

Para calcular finalmente R , G y B :

$$R = 255 \times (r + m) \quad G = 255 \times (g + m) \quad B = 255 \times (b + m)$$

Donde los valores de salida son R, G y B :

$$0 \leq R \leq 255 \quad 0 \leq G \leq 255 \quad 0 \leq B \leq 255$$

Ejemplo de Conversión de HSV a RGB

Para la imagen dada HSV , convertirla a RGB :

$$H = \begin{bmatrix} 0^\circ & 120^\circ \\ 240^\circ & 0^\circ \end{bmatrix} \quad S = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad V = \begin{bmatrix} 1 & 1 \\ 1 & 0.498 \end{bmatrix}$$

Los valores de entrada H , S y V deben estar entre:

$$0^\circ \leq H \leq 360^\circ \quad 0 \leq S \leq 1 \quad 0 \leq V \leq 1$$

Solución

Para el cálculo de C , X , m , r , g y b de cada pixel se hace con:

$$C = V \times S \quad X = C \times \left(1 - \left| \text{mod2}\left(\frac{H}{60^\circ}\right) - 1 \right| \right) \quad m = V - C$$

El término $(1 - |mod2(\frac{H}{60}) - 1|)$ se utiliza en la conversión para encontrar la posición relativa dentro de un ciclo de 120° grados. Al restar 1 y tomar el valor absoluto, este varía entre 0 y 1 y vuelve a 0 cada 120° grados, asegurando que los colores intermedios se calculen adecuadamente. Finalmente, al restar el resultado de 1, se obtiene el valor para ajustar las componentes *RGB* según la posición del matiz dentro del espectro de colores.

$$(r, g, b) = \begin{cases} (C, X, 0) & \text{si } 0^\circ \leq H < 60^\circ \\ (X, C, 0) & \text{si } 60^\circ \leq H < 120^\circ \\ (0, C, X) & \text{si } 120^\circ \leq H < 180^\circ \\ (0, X, C) & \text{si } 180^\circ \leq H < 240^\circ \\ (X, 0, C) & \text{si } 240^\circ \leq H < 300^\circ \\ (C, 0, X) & \text{si } 300^\circ \leq H < 360^\circ \end{cases}$$

Seguidamente, calcular *R*, *G* y *B*:

$$R = 255 \times (r + m) \quad G = 255 \times (g + m) \quad B = 255 \times (b + m)$$

La Tabla 5.2 resume los cálculos numéricos de *C*, *X*, *m*, *r*, *g*, *b* de cada pixel (i, j) :

Pixel	<i>H</i>	<i>S</i>	<i>V</i>	<i>C</i>	<i>X</i>	<i>m</i>	<i>r</i>	<i>g</i>	<i>b</i>	<i>R</i>	<i>G</i>	<i>B</i>
(1, 1)	0	1	1	1	0	0	1	0	0	255	0	0
(1, 2)	120	1	1	1	1	0	0	1	0	0	255	0
(2, 1)	240	1	1	1	0	0	0	0	1	0	0	255
(2, 2)	0	0	0.498	0	0	0.498	0	0	0	127	127	127

Cuadro 5.2: Cálculos numéricos de *C*, *X*, *m*, *r*, *g*, *b*, *R*, *G* y *B* para cada pixel.

El resultado en formato *RGB* es:

$$R = \begin{bmatrix} 255 & 0 \\ 0 & 127 \end{bmatrix}, \quad G = \begin{bmatrix} 0 & 255 \\ 0 & 127 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 255 & 127 \end{bmatrix}$$

5.2.9. Conversión de *RGB* a *XYZ*

Los valores de entrada son *R*, *G* y *B*:

$$0 \leq R \leq 255 \quad 0 \leq G \leq 255 \quad 0 \leq B \leq 255$$

La imagen deben normalizarse entre 0 y 1, por lo que se dividen entre 255, así:

$$r = \frac{R}{255} \quad g = \frac{G}{255} \quad b = \frac{B}{255}$$

Los valores *RGB* normalizados se denotan con $V(r, g, b)$. La misma operación se realiza en los tres canales por separado, luego la función de compresión asociada se usa para obtener los valores $v(r_l, g_l, b_l)$ con corrección de gamma.

$$v = \begin{cases} V/12.92 & \text{si } V \leq 0.04045 \\ \left(\frac{V+0.055}{1.055}\right)^{2.4} & \text{en otro caso} \end{cases}$$

Después se utiliza la matriz M de transformación, bajo un iluminante $D65$, para llegar desde los valores lineales $v(r_l, g_l, b_l)$ a XYZ :

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [M] \begin{bmatrix} r_l \\ g_l \\ b_l \end{bmatrix} \quad \text{donde} \quad M = \begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix}$$

Obtenida la matriz de valores X , Y y Z , los valores normalizados X_n , Y_n y Z_n se calculan como:

$$X_n = \frac{X}{X + Y + Z} \quad Y_n = \frac{Y}{X + Y + Z} \quad Z_n = \frac{Z}{X + Y + Z}$$

Donde los resultados de la salida normalizados son X_n , Y_n y Z_n :

$$0 \leq X_n \leq 1 \quad 0 \leq Y_n \leq 1 \quad 0 \leq Z_n \leq 1$$

Ejemplo de conversión RGB a XYZ

Dadas las matrices de colores R , G y B , pasarlas a X , Y y Z :

$$R = \begin{bmatrix} 255 & 0 \\ 0 & 127 \end{bmatrix} \quad G = \begin{bmatrix} 0 & 255 \\ 0 & 127 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 255 & 127 \end{bmatrix}$$

Solución

Usando la versión normalizada de R , G y B :

$$r = \begin{bmatrix} 1 & 0 \\ 0 & 0.498 \end{bmatrix} \quad g = \begin{bmatrix} 0 & 1 \\ 0 & 0.498 \end{bmatrix} \quad b = \begin{bmatrix} 0 & 0 \\ 1 & 0.498 \end{bmatrix}$$

Los canales RGB normalizados se denotan con $V(r, g, b)$, para así calcular las matrices $v(r_l, g_l, b_l)$ con la corrección de gamma:

$$v(r_l, g_l, b_l) = \begin{cases} \frac{V}{12.92} & \text{si } V \leq 0.04045 \\ \left(\frac{V+0.055}{1.055}\right)^{2.4} & \text{en otro caso} \end{cases}$$

Así, $v(r_l, g_l, b_l)$ después de la corrección gamma:

$$r_l = \begin{bmatrix} \left(\frac{1+0.055}{1.055}\right)^{2.4} & 0 \\ 0 & \left(\frac{0.498+0.055}{1.055}\right)^{2.4} \end{bmatrix} \quad g_l = \begin{bmatrix} 0 & \left(\frac{1+0.055}{1.055}\right)^{2.4} \\ 0 & \left(\frac{0.498+0.055}{1.055}\right)^{2.4} \end{bmatrix} \quad b_l = \begin{bmatrix} 0 & 0 \\ \left(\frac{1+0.055}{1.055}\right)^{2.4} & \left(\frac{0.498+0.055}{1.055}\right)^{2.4} \end{bmatrix}$$

$$r_l = \begin{bmatrix} 1 & 0 \\ 0 & 0.2158 \end{bmatrix} \quad g_l = \begin{bmatrix} 0 & 1 \\ 0 & 0.2158 \end{bmatrix} \quad b_l = \begin{bmatrix} 0 & 0 \\ 1 & 0.2158 \end{bmatrix}$$

Utilizando la matriz M de transformación para $D65$ a cada pixel:

$$\begin{bmatrix} X(i, j) \\ Y(i, j) \\ Z(i, j) \end{bmatrix} = \begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix} \begin{bmatrix} r_l(i, j) \\ g_l(i, j) \\ b_l(i, j) \end{bmatrix}$$

De manera ilustrativa, para $i = 1, j = 1$:

$$X(1,1) = 0.4124564 \times 1 + 0.3575761 \times 0 + 0.1804375 \times 0 = 0.4125$$

$$Y(1,1) = 0.2126729 \times 1 + 0.7151522 \times 0 + 0.0721750 \times 0 = 0.2127$$

$$Z(1,1) = 0.0193339 \times 1 + 0.1191920 \times 0 + 0.9503041 \times 0 = 0.0193$$

Obtenidos X , Y y Z , los valores normalizados X_n , Y_n y Z_n se calculan como se aprecia en el ejemplo para $(1,1)$:

$$X_n(1,1) = \frac{X(1,1)}{X(1,1) + Y(1,1) + Z(1,1)} = \frac{0.4125}{0.4125 + 0.2127 + 0.0193} = 0.6392$$

$$Y_n(1,1) = \frac{Y(1,1)}{X(1,1) + Y(1,1) + Z(1,1)} = \frac{0.2127}{0.4125 + 0.2127 + 0.0193} = 0.3296$$

$$Z_n(1,1) = \frac{Z(1,1)}{X(1,1) + Y(1,1) + Z(1,1)} = \frac{0.0193}{0.4125 + 0.2127 + 0.0193} = 0.0299$$

La Tabla 5.3 resume los cálculos numéricos de $r, g, b, r_l, g_l, b_l, X, Y, Z, X_n, Y_n, Z_n$, de cada pixel (i, j) :

Pixel	r	g	b	r_l	g_l	b_l	X	Y	Z	X_n	Y_n	Z_n
(1,1)	1	0	0	1	0	0	0.4125	0.2127	0.0193	0.6392	0.3296	0.0299
(1,2)	0	1	0	0	1	0	0.3576	0.7152	0.1192	0.3000	0.6000	0.1000
(2,1)	0	0	1	0	0	1	0.1804	0.0722	0.9503	0.1500	0.0600	0.7900
(2,2)	0.4980	0.4980	0.4980	0.2158	0.2158	0.2158	0.2017	0.2122	0.2311	0.3127	0.3290	0.3583

Cuadro 5.3: Cálculos numéricos de $r, g, b, r_l, g_l, b_l, X, Y, Z, X_n, Y_n, Z_n$, de cada pixel dela conversión de RGB a XYZ.

El resultado en formato XYZ no normalizado es:

$$X = \begin{bmatrix} 0.4125 & 0.3576 \\ 0.1804 & 0.2017 \end{bmatrix} \quad Y = \begin{bmatrix} 0.2127 & 0.7152 \\ 0.0722 & 0.2122 \end{bmatrix} \quad Z = \begin{bmatrix} 0.0193 & 0.1192 \\ 0.9503 & 0.2311 \end{bmatrix}$$

5.2.10. Conversión de XYZ a RGB

Los valores de entrada son X , Y y Z los cuales no están normalizados. La función $V(r, g, b)$ se obtiene aplicando la matriz de transformación D65:

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = [M]^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \text{donde} \quad M^{-1} = \begin{bmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{bmatrix}$$

La misma operación se realiza en los tres canales por separado, pero depende de la función de compresión asociada para obtener los valores de $v(r_l, g_l, b_l)$:

$$v(r_l, g_l, b_l) = \begin{cases} 12.92V & \text{si } V \leq 0.00313081 \\ 1.055V^{1/2.4} - 0.055 & \text{en otro caso} \end{cases}$$

Para calcular R , G y B :

$$R = 255 \times r_l \quad G = 255 \times g_l \quad B = 255 \times b_l$$

Los valores de salida son R , G y B :

$$0 \leq R \leq 255 \quad 0 \leq G \leq 255 \quad 0 \leq B \leq 255$$

Ejemplo de conversión XYZ a RGB

Dadas las matrices de entrada de colores X , Y y Z , pasarlitas a R, G y B :

$$X = \begin{bmatrix} 0.4125 & 0.3576 \\ 0.1804 & 0.2017 \end{bmatrix} \quad Y = \begin{bmatrix} 0.2127 & 0.7152 \\ 0.0722 & 0.2122 \end{bmatrix} \quad Z = \begin{bmatrix} 0.0193 & 0.1192 \\ 0.9503 & 0.2311 \end{bmatrix}$$

Solución

La función $V(r, g, b)$ se obtiene aplicando la matriz de transformación:

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = [M]^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \text{donde} \quad M^{-1} = \begin{bmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{bmatrix}$$

Aplicando a cada pixel (i,j):

$$\begin{bmatrix} r(i, j) \\ g(i, j) \\ b(i, j) \end{bmatrix} = \begin{bmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{bmatrix} \begin{bmatrix} X(i, j) \\ Y(i, j) \\ Z(i, j) \end{bmatrix}$$

De manera ilustrativa, para $i = 1, j = 1$:

$$\begin{bmatrix} r(1, 1) \\ g(1, 1) \\ b(1, 1) \end{bmatrix} = \begin{bmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{bmatrix} \cdot \begin{bmatrix} 0.4125 \\ 0.2127 \\ 0.0193 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}$$

Las matrices r , g y b con los resultados calculados son:

$$r = \begin{bmatrix} 1.0000 & 0.0000 \\ 0.0000 & 0.2123 \end{bmatrix} \quad g = \begin{bmatrix} 0.0000 & 1.0000 \\ 0.0000 & 0.2123 \end{bmatrix} \quad b = \begin{bmatrix} 0.0000 & 0.0000 \\ 1.0000 & 0.2123 \end{bmatrix}$$

Seguidamente, se evalúa la función de compresión asociada para obtener los valores de $v(r_l, g_l, b_l)$:

$$v(r_l, g_l, b_l) = \begin{cases} 12.92V & \text{si } V \leq 0.00313081 \\ 1.055V^{1/2.4} - 0.055 & \text{en otro caso} \end{cases}$$

Para los canales r_l , g_l y b_l :

$$r_l = \begin{bmatrix} 1.055 \times 1.0000^{1/2.4} - 0.055 & 12.92 \times 0.0000 \\ 12.92 \times 0.0000 & 1.055 \times 0.2123^{1/2.4} - 0.055 \end{bmatrix} \quad g_l = \begin{bmatrix} 12.92 \times 0.0000 & 1.055 \times 1.0000^{1/2.4} - 0.055 \\ 12.92 \times 0.0000 & 1.055 \times 0.2122^{1/2.4} - 0.055 \end{bmatrix}$$

$$b_l = \begin{bmatrix} 12.92 \times 0.0000 & 12.92 \times 0.0000 \\ 1.055 \times 1.0000^{1/2.4} - 0.055 & 1.055 \times 0.2123^{1/2.4} - 0.055 \end{bmatrix}$$

$$r_l = \begin{bmatrix} 1.0000 & 0.0000 \\ 0.0000 & 0.4980 \end{bmatrix} \quad g_l = \begin{bmatrix} 0.0000 & 1.0000 \\ 0.0000 & 0.4980 \end{bmatrix} \quad b_l = \begin{bmatrix} 0.0000 & 0.0000 \\ 1.0000 & 0.4980 \end{bmatrix}$$

Las matrices R , G y B se calculan como:

$$R = 255 \times r_l \quad G = 255 \times g_l \quad B = 255 \times b_l$$

$$R = 255 \times \begin{bmatrix} 1.0000 & 0.0000 \\ 0.0000 & 0.4980 \end{bmatrix} \quad G = 255 \times \begin{bmatrix} 0.0000 & 1.0000 \\ 0.0000 & 0.4980 \end{bmatrix} \quad B = 255 \times \begin{bmatrix} 0.0000 & 0.0000 \\ 1.0000 & 0.4980 \end{bmatrix}$$

La Tabla 5.4 resume los cálculos numéricos de $X, Y, Z, r, g, b, r_l, g_l, b_l, R, G, B$ de cada pixel (i, j) :

Pixel	X	Y	Z	r	g	b	r_l	g_l	b_l	R	G	B
(1,1)	0.4125	0.2127	0.0193	1.0000	0.0000	0.0000	1.0000	0.0000	0.0000	255	0	0
(1,2)	0.3576	0.7152	0.1192	0.0000	1.0000	0.0000	0.0000	1.0000	0.0000	0	255	0
(2,1)	0.1804	0.0722	0.9503	0.0000	0.0000	1.0000	0.0000	0.0000	1.0000	0	0	255
(2,2)	0.2017	0.2122	0.2311	0.2123	0.2123	0.2123	0.4980	0.4980	0.4980	127	127	127

Cuadro 5.4: Cálculos numéricos de $X, Y, Z, r, g, b, r_l, g_l, b_l, R, G, B$, de cada pixel de la conversión de XYZ a RGB.

Una vez multiplicado y redondeado a entero, el resultado en formato RGB es:

$$R = \begin{bmatrix} 255 & 0 \\ 0 & 127 \end{bmatrix} \quad G = \begin{bmatrix} 0 & 255 \\ 0 & 127 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 255 & 127 \end{bmatrix}$$

Los resultados de salida son R , G y B cumplen con:

$$0 \leq R \leq 255 \quad 0 \leq G \leq 255 \quad 0 \leq B \leq 255$$

5.2.11. Conversión de XYZ a Lab

Los valores de entrada son X , Y y Z no normalizados. Esta conversión requiere el punto blanco de referencia (X_r, Y_r, Z_r) el cual se selecciona partir de la Tabla 5.5 y así obtener x_r , y_r , y z_r :

	Estándar	X_r	Y_r	Z_r
d50		0.9642	1	0.8251
d55		0.9568	1	0.9214
d65 *		0.9504	1	1.0888
ICC		0.9642	1	0.8249

Cuadro 5.5: Valores de referencia para el punto blanco en diferentes estándares. Los marcados con * son los frecuentemente usados.

$$x_r = \frac{X}{X_r} \quad y_r = \frac{Y}{Y_r} \quad z_r = \frac{Z}{Z_r}$$

Posteriormente, se evalúan $v(x_r, y_r, z_r)$ en la función f_v para cada valor y así obtener f_x , f_y , y f_z , respectivamente:

$$f_v = \begin{cases} \sqrt[3]{v_r} & \text{si } v_r > \epsilon \\ \kappa v_r + \frac{16}{116} & \text{en otro caso} \end{cases}$$

Se definen las constantes κ y ϵ utilizadas en las condiciones de la conversión, definidas en la Tabla 5.6:

Constante	Actual estándar de la CIE	Intento del estándar de la CIE*
ϵ	0.008856	$\frac{216}{24389}$
κ	903.3	$\frac{24389}{27}$

Cuadro 5.6: Constantes ϵ y κ utilizadas en las condiciones de la conversión. Los valores marcados con * son lo usados usualmente.

Las componentes L , a y b se calculan como:

$$L = 116f_y - 16 \quad a = 500(f_x - f_y) \quad b = 200(f_y - f_z)$$

Los valores de salida son L , a y b :

$$0 \leq L \leq 100 \quad -100 \leq a \leq 100 \quad -100 \leq b \leq 100$$

Nota: los valores marcados con * son lo usados usualmente.

Ejemplo de conversión de XYZ a Lab

Las Matrices de entrada XYZ:

$$X = \begin{bmatrix} 0.4125 & 0.3576 \\ 0.1804 & 0.2017 \end{bmatrix} \quad Y = \begin{bmatrix} 0.2127 & 0.7152 \\ 0.0722 & 0.2122 \end{bmatrix} \quad Z = \begin{bmatrix} 0.0193 & 0.1192 \\ 0.9503 & 0.2311 \end{bmatrix}$$

Solución

Se definen los valores de referencia del punto blanco (usando $D65$) leídos de la Tabla 5.5 :

$$X_r = 0.95047, \quad Y_r = 1.00000, \quad Z_r = 1.08883$$

Luego se calcula x_r, y_r, z_r , así:

$$x_r = \frac{X}{X_r}, \quad y_r = \frac{Y}{Y_r}, \quad z_r = \frac{Z}{Z_r}$$

$$x_r = \begin{bmatrix} 0.4340 & 0.3762 \\ 0.1898 & 0.2122 \end{bmatrix} \quad y_r = \begin{bmatrix} 0.2127 & 0.7152 \\ 0.0722 & 0.2122 \end{bmatrix} \quad z_r = \begin{bmatrix} 0.0177 & 0.1095 \\ 0.8728 & 0.2122 \end{bmatrix}$$

Seguidamente, se definen las constantes ϵ y κ usando la Tabla 5.6:

$$\epsilon = 0.008856, \quad \kappa = 903.3$$

Se calculan f_x, f_y, f_z usando las condiciones dadas:

$$f_x = \begin{cases} \sqrt[3]{x_r} & \text{si } x_r > \epsilon \\ \kappa x_r + \frac{16}{116} & \text{en otro caso} \end{cases} \quad f_y = \begin{cases} \sqrt[3]{y_r} & \text{si } y_r > \epsilon \\ \kappa y_r + \frac{16}{116} & \text{en otro caso} \end{cases} \quad f_z = \begin{cases} \sqrt[3]{z_r} & \text{si } z_r > \epsilon \\ \kappa z_r + \frac{16}{116} & \text{en otro caso} \end{cases}$$

Así:

$$f_x = \begin{bmatrix} \sqrt[3]{0.4340} & \sqrt[3]{0.3762} \\ \sqrt[3]{0.1898} & \sqrt[3]{0.2122} \end{bmatrix} \quad f_y = \begin{bmatrix} \sqrt[3]{0.2127} & \sqrt[3]{0.7152} \\ \sqrt[3]{0.0722} & \sqrt[3]{0.2122} \end{bmatrix} \quad f_z = \begin{bmatrix} \sqrt[3]{0.0177} & \sqrt[3]{0.1095} \\ \sqrt[3]{0.8728} & \sqrt[3]{0.2122} \end{bmatrix}$$

Aplicadas la condiciones:

$$f_x = \begin{bmatrix} 0.7564 & 0.7215 \\ 0.5755 & 0.5956 \end{bmatrix} \quad f_y = \begin{bmatrix} 0.5949 & 0.8945 \\ 0.4124 & 0.5945 \end{bmatrix} \quad f_z = \begin{bmatrix} 0.2600 & 0.4782 \\ 0.9546 & 0.5956 \end{bmatrix}$$

Paso 5: Calcular L, a, b:

$$L = 116f_y - 16 \quad L = 116 \begin{bmatrix} 0.5949 & 0.8945 \\ 0.4124 & 0.5945 \end{bmatrix} - 16$$

$$a = 500(f_x - f_y) \quad a = 500 \left(\begin{bmatrix} 0.7564 & 0.7215 \\ 0.5755 & 0.5956 \end{bmatrix} - \begin{bmatrix} 0.5949 & 0.8945 \\ 0.4124 & 0.5945 \end{bmatrix} \right)$$

$$b = 200(f_y - f_z) \quad b = 200 \left(\begin{bmatrix} 0.5949 & 0.8945 \\ 0.4124 & 0.5945 \end{bmatrix} - \begin{bmatrix} 0.2600 & 0.4782 \\ 0.9546 & 0.5956 \end{bmatrix} \right)$$

Resultado final para L, a y b :

$$L = \begin{bmatrix} 53.2084 & 87.7620 \\ 31.8384 & 52.9620 \end{bmatrix} \quad a = \begin{bmatrix} 80.7500 & -86.5000 \\ 81.5500 & 0.5500 \end{bmatrix} \quad b = \begin{bmatrix} 66.9800 & 83.2600 \\ -108.4400 & -0.2200 \end{bmatrix}$$

La Tabla 5.7 resume los cálculos numéricos de $X, Y, Z, x_r, y_r, z_r, f_x, f_y, f_z, L, a, b$ de cada pixel (i, j) :

Pixel	X	Y	Z	x_r	y_r	z_r	f_x	f_y	f_z	L	a	b
(1,1)	0.4125	0.2127	0.0193	0.4340	0.2127	0.0177	0.7564	0.5949	0.2600	53.2084	80.7500	66.9800
(1,2)	0.3576	0.7152	0.1192	0.3762	0.7152	0.1095	0.7215	0.8945	0.4782	87.7620	-86.5000	83.2600
(2,1)	0.1804	0.0722	0.9503	0.1898	0.0722	0.8728	0.5755	0.4124	0.9546	31.8384	81.5500	-108.4400
(2,2)	0.2017	0.2122	0.2311	0.2122	0.2122	0.2122	0.5956	0.5945	0.5956	52.9620	0.5500	-0.2200

Cuadro 5.7: Cálculos numéricos de $X, Y, Z, x_r, y_r, z_r, f_x, f_y, f_z, L, a, b$ de cada pixel de la conversión XYZ a Lab .

Estos valores cumplen con las condiciones de salida:

$$0 \leq L \leq 100 \quad -100 \leq a \leq 100 \quad -100 \leq b \leq 100$$

5.2.12. Conversión de Lab a XYZ

Los valores de entrada son L, a y b :

$$0 \leq L \leq 100 \quad -1 \leq a \leq 100 \quad -100 \leq b \leq 100$$

Se calculan las componentes f_x, f_y y f_z :

$$f_y = \frac{L + 16}{116} \quad f_x = \frac{a}{500} + f_y \quad f_z = f_y - \frac{b}{200}$$

Se definen las constantes κ y ϵ utilizadas en las condiciones de la conversión en la Tabla 5.6:

Se calculan las componentes X_r, Y_r y Z_r aplicando las condiciones según las fórmulas proporcionadas:

$$x_r = \begin{cases} f_x^3 & \text{si } f_x^3 > \epsilon \\ \frac{116f_x - 16}{\kappa} & \text{en otro caso} \end{cases} \quad y_r = \begin{cases} \left(\frac{L+16}{116}\right)^3 & \text{si } L > \kappa\epsilon \\ \frac{L}{\kappa} & \text{en otro caso} \end{cases} \quad z_r = \begin{cases} f_z^3 & \text{si } f_z^3 > \epsilon \\ \frac{116f_z - 16}{\kappa} & \text{en otro caso} \end{cases}$$

Para evaluar X, Y y Z se usan los valores de referencia para el punto blanco:

$$X = x_r \cdot X_r \quad Y = y_r \cdot Y_r \quad Z = z_r \cdot Z_r$$

Los valores de salida son X, Y y Z no normalizados.

Ejemplo de conversión de Lab a XYZ

Matrices de entrada Lab:

$$L = \begin{bmatrix} 53.2084 & 87.7620 \\ 31.8384 & 52.9620 \end{bmatrix} \quad a = \begin{bmatrix} 80.7500 & -86.5000 \\ 81.5500 & 0.5500 \end{bmatrix} \quad b = \begin{bmatrix} 66.9800 & 83.2600 \\ -108.4400 & -0.2200 \end{bmatrix}$$

Solución

Calcular f_y, f_x, f_z :

$$f_y = \frac{L + 16}{116} \quad f_x = \frac{a}{500} + f_y \quad f_z = f_y - \frac{b}{200}$$

$$f_y = \begin{bmatrix} \frac{53.2084+16}{116} & \frac{87.7620+16}{116} \\ \frac{31.8384+16}{116} & \frac{52.9620+16}{116} \end{bmatrix} \quad f_x = \begin{bmatrix} \frac{80.7500}{500} + 0.5949 & \frac{-86.5000}{500} + 0.8945 \\ \frac{81.5500}{500} + 0.4124 & \frac{0.5500}{500} + 0.5945 \end{bmatrix} \quad f_z = \begin{bmatrix} 0.5949 - \frac{66.9800}{200} & 0.8945 - \frac{83.2600}{200} \\ 0.4124 - \frac{-108.4400}{200} & 0.5945 - \frac{-0.2200}{200} \end{bmatrix}$$

$$f_y = \begin{bmatrix} 0.5949 & 0.8945 \\ 0.4124 & 0.5945 \end{bmatrix} \quad f_x = \begin{bmatrix} 0.7564 & 0.7215 \\ 0.5755 & 0.5956 \end{bmatrix} \quad f_z = \begin{bmatrix} 0.2600 & 0.4782 \\ 0.9546 & 0.5956 \end{bmatrix}$$

Seguidamente, se definen las constantes ϵ y κ usando la Tabla 5.6:

$$\epsilon = 0.008856, \quad \kappa = 903.3$$

Para calcular x_r, y_r, z_r

$$x_r = \begin{cases} f_x^3 & \text{si } f_x^3 > \epsilon \\ \frac{116f_x - 16}{\kappa} & \text{en otro caso} \end{cases} \quad y_r = \begin{cases} \left(\frac{L+16}{116}\right)^3 & \text{si } \left(\frac{L+16}{116}\right)^3 > \epsilon \\ \frac{L}{\kappa} & \text{en otro caso} \end{cases} \quad z_r = \begin{cases} f_z^3 & \text{si } f_z^3 > \epsilon \\ \frac{116f_z - 16}{\kappa} & \text{en otro caso} \end{cases}$$

Por lo tanto:

$$x_r = \begin{bmatrix} 0.7564^3 & 0.7215^3 \\ 0.5755^3 & 0.5956^3 \end{bmatrix} \quad y_r = \begin{bmatrix} 0.5949^3 & 0.8945^3 \\ 0.4124^3 & 0.5945^3 \end{bmatrix} \quad z_r = \begin{bmatrix} 0.2600^3 & 0.4782^3 \\ 0.9546^3 & 0.5956^3 \end{bmatrix}$$

Los resultados finales:

$$x_r = \begin{bmatrix} 0.4332 & 0.3756 \\ 0.1904 & 0.2115 \end{bmatrix} \quad y_r = \begin{bmatrix} 0.2103 & 0.7152 \\ 0.0700 & 0.2103 \end{bmatrix} \quad z_r = \begin{bmatrix} 0.0176 & 0.1094 \\ 0.8689 & 0.2115 \end{bmatrix}$$

Usando los valores de referencia del punto blanco (D65) se obtienen X, Y y Z :

$$X_r = 0.95047, \quad Y_r = 1.00000, \quad Z_r = 1.08883$$

$$X = x_r \cdot X_r \quad Y = y_r \cdot Y_r \quad Z = z_r \cdot Z_r$$

La Tabla 5.8 resume los cálculos numéricos de $L, a, b, f_y, f_x, f_z, x_r, y_r, z_r, X, Y, Z$, de cada pixel:

Pixel	L	a	b	f_y	f_x	f_z	x_r	y_r	z_r	X	Y	Z
(1,1)	53.2084	80.7500	66.9800	0.5949	0.7564	0.2600	0.4332	0.2103	0.0176	0.4118	0.2103	0.0192
(1,2)	87.7620	-86.5000	83.2600	0.8945	0.7215	0.4782	0.3756	0.7152	0.1094	0.3570	0.7152	0.1191
(2,1)	31.8384	81.5500	-108.4400	0.4124	0.5755	0.9546	0.1904	0.0700	0.8689	0.1810	0.0700	0.9461
(2,2)	52.9620	0.5500	-0.2200	0.5945	0.5956	0.5956	0.2115	0.2103	0.2115	0.2010	0.2103	0.2303

Cuadro 5.8: Cálculos numéricos de $L, a, b, f_y, f_x, f_z, x_r, y_r, z_r, X, Y, Z$, de cada pixel de la conversión de *Lab* a *XYZ*.

Los valores de salida finales son X, Y y Z no normalizados.

$$X = \begin{bmatrix} 0.4118 & 0.3570 \\ 0.1810 & 0.2010 \end{bmatrix} \quad Y = \begin{bmatrix} 0.2103 & 0.7152 \\ 0.0700 & 0.2103 \end{bmatrix} \quad Z = \begin{bmatrix} 0.0192 & 0.1191 \\ 0.9461 & 0.2303 \end{bmatrix}$$

La diferencia entre los valores originales de *XYZ* y los obtenidos después de la conversión es común y se debe a varias razones relacionadas con redondeos y aproximaciones durante los cálculos. Durante las operaciones numéricas, como la exponenciación y la división, los valores pueden redondearse según el entorno de cálculo usado, lo que introduce algunas discrepancias. Además, las aproximaciones en las constantes utilizadas en las fórmulas de conversión, como ϵ y κ , también contribuyen a las diferencias, ya que no representan valores exactos.

5.2.13. Conversión de Lab a RGB

Para convertir una imagen del espacio de color LAB al espacio de color RGB, generalmente se pasa de LAB al espacio de color CIE XYZ. Esto se hace utilizando las ecuaciones previamente definidas en 5.2.12. Posteriormente, se convierte de XYZ al espacio de color RGB como se muestra en 5.2.10. Los detalles exactos de las ecuaciones y matrices de conversión, como ya se ha indicado, pueden variar dependiendo de la implementación específica del algoritmo, por lo que deben usarse la misma manera mantener la consistencia.

Ejemplo de conversión de Lab a RGB

Convertir de *Lab* a *RGB*:

$$L = \begin{bmatrix} 53.2084 & 87.7620 \\ 31.8384 & 52.9620 \end{bmatrix} \quad a = \begin{bmatrix} 80.7500 & -86.5000 \\ 81.5500 & 0.5500 \end{bmatrix} \quad b = \begin{bmatrix} 66.9800 & 83.2600 \\ -108.4400 & -0.2200 \end{bmatrix}$$

Solución

Primero se debe convertir de *Lab* a *XYZ*, como se realizó en la sección 5.2.12:

$$X = \begin{bmatrix} 0.4118 & 0.3570 \\ 0.1810 & 0.2010 \end{bmatrix} \quad Y = \begin{bmatrix} 0.2103 & 0.7152 \\ 0.0700 & 0.2103 \end{bmatrix} \quad Z = \begin{bmatrix} 0.0192 & 0.1191 \\ 0.9461 & 0.2303 \end{bmatrix}$$

Seguidamente, este resultado se lleva desde *XYZ* a *RGB* como se indicó en la sección 5.2.10:

$$R = \begin{bmatrix} 255 & 0 \\ 0 & 127 \end{bmatrix} \quad G = \begin{bmatrix} 0 & 255 \\ 0 & 127 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 255 & 127 \end{bmatrix}$$

5.2.14. Conversión de RGB a Lab

Para convertir una imagen de *RGB* al espacio de color *Lab*, se aplica un algoritmo que inicialmente lleva del *RGB* a *XYZ* usando el método presentado en 5.2.9. Luego, los valores *XYZ* se transforman en coordenadas *Lab* utilizando

las ecuaciones definidas en [5.2.11](#).

Ejemplo de conversión de RGB a Lab

Convertir de *RGB* a *Lab*:

$$R = \begin{bmatrix} 255 & 0 \\ 0 & 127 \end{bmatrix} \quad G = \begin{bmatrix} 0 & 255 \\ 0 & 127 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 255 & 127 \end{bmatrix}$$

Solución

Primero se debe convertir de *RGB* a *XYZ*, como se realizó en la sección [5.2.9](#):

$$X = \begin{bmatrix} 0.4118 & 0.3570 \\ 0.1810 & 0.2010 \end{bmatrix} \quad Y = \begin{bmatrix} 0.2103 & 0.7152 \\ 0.0700 & 0.2103 \end{bmatrix} \quad Z = \begin{bmatrix} 0.0192 & 0.1191 \\ 0.9461 & 0.2303 \end{bmatrix}$$

Seguidamente, este resultado se lleva desde *XYZ* a *Lab* como se indicó en la sección [5.2.11](#):

$$L = \begin{bmatrix} 53.2084 & 87.7620 \\ 31.8384 & 52.9620 \end{bmatrix} \quad a = \begin{bmatrix} 80.7500 & -86.5000 \\ 81.5500 & 0.5500 \end{bmatrix} \quad b = \begin{bmatrix} 66.9800 & 83.2600 \\ -108.4400 & -0.2200 \end{bmatrix}$$

5.3. Ejemplo Segmentación

Para la imagen de Cartagena se propone realizar la segmentación de color en los modelos *RGB*, *HSV* y *CIELAB*.

Solución

La Figura [5.10](#) muestra el proceso de separación del color amarillo de la bandera de la imagen ya usada de Cartagena usando *RGB*. En ella, es necesario segmentar algunas intensidades sobre las capas de rojo, verde y azul. Es un modelo poco intuitivo y el color es la resultante de la mezcla de varias tonalidades. En [\(5.2\)](#) se presentan los umbrales usados para la segmentación de color amarillo usando el modelo *RGB*. Los valores de las tres capas están comprendidos entre 0 y 255 siendo siempre valores enteros.



Figura 5.10: Segmentación del color amarillo de la bandera de Colombia en la imagen de Cartagena usando *RGB*.

$$\begin{aligned} 130 \leq R \leq 185 \\ 63 \leq G \leq 165 \\ 0 \leq B \leq 45 \end{aligned} \quad (5.2)$$

La Figura 5.11 muestra el proceso de separación del color amarillo de la bandera usando el modelo del color HSV. Para segmentar el color amarillo es primordial el uso de la capa de la tonalidad H. La saturación S y el valor V debe ser modificar para definir un mejor ajuste de la separación de la bandera en su parte amarilla. En (5.3) se presentan los umbrales usados para la segmentación de color amarillo usando el modelo HSV. El valor de la capa H está normalizado entre 0 y 1, no obstante es frecuente encontrarle también entre 0 y 360 grados por ser un disco. Las capas S y V son valores entre 0 y 1.

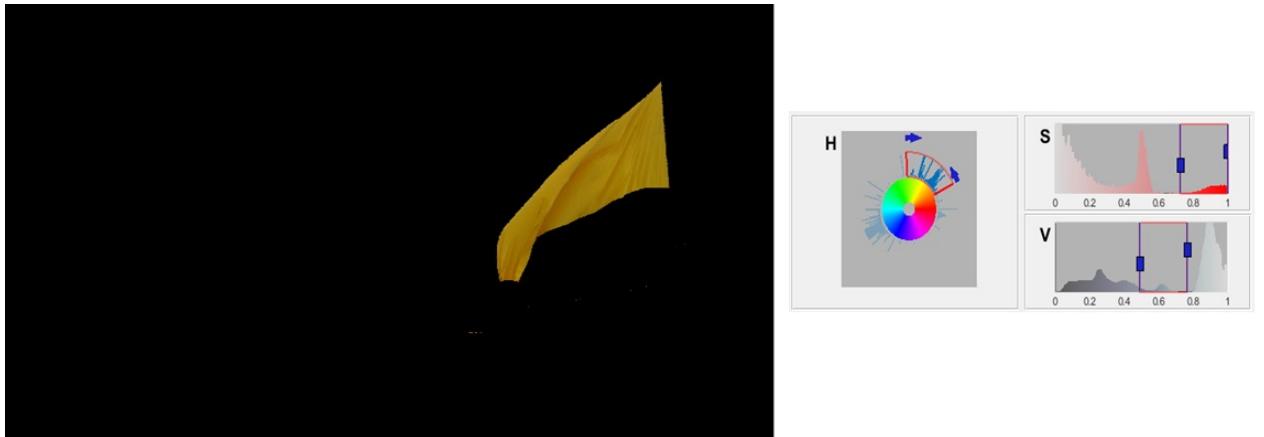


Figura 5.11: Segmentación del color amarillo de la bandera de Colombia en la imagen de Cartagena usando HSV.

$$\begin{aligned} 0 \leq H \leq 0.27 \\ 0.72 \leq S \leq 1 \\ 0.48 \leq V \leq 0.76 \end{aligned} \quad (5.3)$$

La Figura 5.12 muestra el proceso de separación del color amarillo de la bandera usando el modelo CIELAB. En este caso, el color amarillo se encuentra en la capa b^* en el lado positivo. La segmentación debió ajustarse sutilmente en el canal a^* y L^* para mejorar la separación del color con sus matices. En (5.12) se presentan los umbrales usados para la segmentación de color amarillo usando el modelo CIELAB. Para la capa L^* los valores son enteros entre 0 y 100. Por otro lado, las capas a^* y b^* son valores enteros entre -100 y 100.

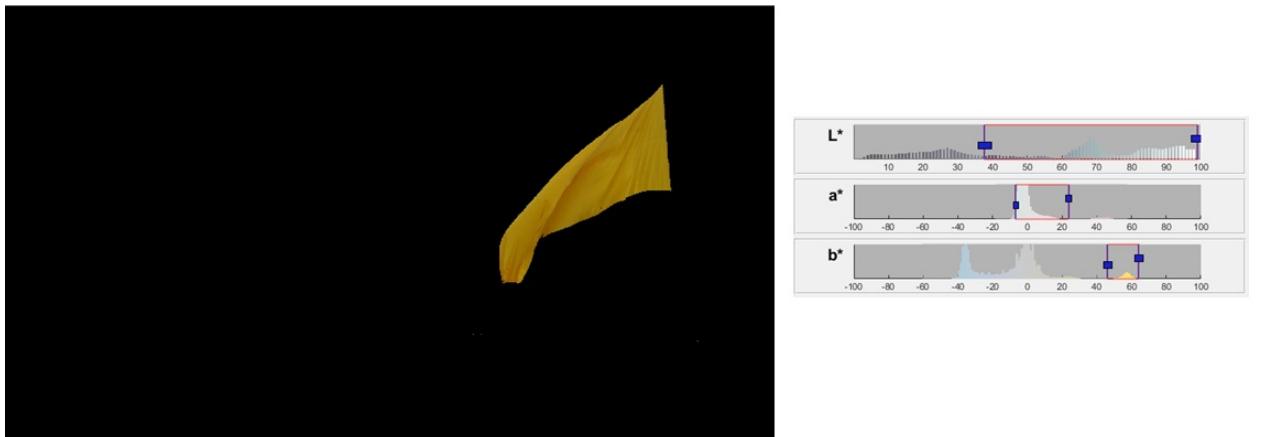


Figura 5.12: Segmentación del color amarillo de la bandera de Colombia en la imagen de Cartagena usando CIELAB.

$$\begin{aligned} 37 &\leq L* \leq 99 \\ -7 &\leq a* \leq 24 \\ 46 &\leq b* \leq 16 \end{aligned} \tag{5.4}$$

5.3.1. Ejemplo en Python: Segmentación de la bandera usando el modelo RGB

Este código realiza la segmentación del color amarillo en Python en la bandera de la imagen 'cartagena.jpg' utilizando el modelo de color RGB. El proceso carga de la imagen y su visualización para identificar las áreas de interés. A seguidamente, se definen los rangos de intensidad en las componentes roja (R), verde (G) y azul (B) que corresponden al color amarillo. Utilizando estos, se genera una máscara binaria que identifica las regiones de la imagen donde los valores cumplen con los criterios establecidos. Luego, esta máscara se aplica a la imagen original para extraer únicamente las regiones amarillas. Finalmente, se visualizan la imagen original, la máscara generada y la imagen segmentada para verificar los resultados

Solución

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import sys
4 from ip_functions import *
5
6 # Abrir imagen
7 RGB=plt.imread("cartagena.jpg")
8
9 #Realizar Slicing para reducir procesamiento
10 S=10
11 RGB=np.array(RGB[1::S,1::S,:])
12
13 """#Extraer Capas RGB"""
14 # RGB es la imagen de entrada
15 r, g, b = imsplit(RGB)
16
17 """#Segmenta RGB"""
18 #Segmentar el color Amarillo
19
20 # Amarillo
21 UmbralInferiorR, UmbralSuperiorR = 130, 185
22 UmbralInferiorG, UmbralSuperiorG = 63, 165
23 UmbralInferiorB, UmbralSuperiorB = 0, 45
24
25 # Crear máscaras
26 MascaraR = (r >= UmbralInferiorR) & (r <= UmbralSuperiorR)
27 MascaraG = (g >= UmbralInferiorG) & (g <= UmbralSuperiorG)
28 MascaraB = (b >= UmbralInferiorB) & (b <= UmbralSuperiorB)
29
30 # Graficar resultados
31 plt.figure(2, figsize=(15, 10))
32 plt.subplot(2, 3, 1)
33 plt.imshow(r, cmap='gray')
34 plt.title('Rojo')
35
36 plt.subplot(2, 3, 2)
37 plt.imshow(g, cmap='gray')
38 plt.title('Verde')
39
40 plt.subplot(2, 3, 3)
41 plt.imshow(b, cmap='gray')
42 plt.title('Azul')
43
44 plt.subplot(2, 3, 4)
45 plt.imshow(MascaraR, cmap='gray')
46 plt.title('Máscara Roja')
47
48 plt.subplot(2, 3, 5)
49 plt.imshow(MascaraG, cmap='gray')
50 plt.title('Máscara Verde')
```

```

51
52 plt.subplot(2, 3, 6)
53 plt.imshow(MascaraB, cmap='gray')
54 plt.title('Máscara Azul')
55
56 plt.tight_layout()
57
58 # Combinar máscaras
59 MascaraFinal = MascaraR & MascaraG & MascaraB
60
61 """#Imagen Segmentada en RGB"""
62 # Graficar resultados
63 plt.figure(2, figsize=(20, 10))
64 plt.subplot(1, 3, 1)
65 plt.imshow(RGB)
66 plt.title('Imagen RGB Original')
67 plt.subplot(1, 3, 2)
68 plt.imshow(MascaraFinal, cmap='gray')
69 plt.title('Máscara Final')
70 plt.subplot(1, 3, 3)
71 plt.imshow(MascaraFinal[:, :, np.newaxis] * RGB)
72 plt.title('Imagen RGB Enmascarada')
73
74 plt.show()

```

La Fig.[5.13](#) muestra la segmentación de la franja amarilla de la bandera de Colombia utilizando el modelo RGB. La parte superior ilustran las capas de cada color, mientras que en el centro se presenta las máscaras binarias generadas para resaltar las áreas correspondientes al color amarillo. En la parte inferior, se muestran la imagen a color, luego la máscara de segmentación y, finalmente, la región amarilla segmentada. Las funciones referidas allí, se encuentran en el capítulo [13](#).



Figura 5.13: Segmentación de la bandera de Colombia en su franja amarilla usando el modelo RGB. En la parte inferior, se muestran la imagen a color, luego la máscara de segmentación y, finalmente, la región amarilla segmentada.

5.3.2. Ejemplo en Python: Segmentación de la bandera usando el modelo HSV

Este código en Python realiza la segmentación del color amarillo en la bandera de la imagen 'cartagena.jpg' utilizando el modelo de color HSV, ofreciendo una solución más intuitiva y efectiva que el modelo RGB. El proceso comienza con la carga de la imagen y su visualización para identificar las áreas de interés. A continuación, se definen rangos en las componentes de tonalidad (H), saturación (S) y valor (V) que corresponden al color amarillo, facilitando una identificación precisa al separar el color de sus atributos de brillo e intensidad. Con estos rangos, se genera una máscara binaria que resalta las regiones de la imagen que cumplen con los criterios establecidos. Posteriormente, esta máscara se aplica a la imagen original para extraer únicamente las áreas correspondientes al color amarillo. Finalmente, se muestran la imagen original, la máscara generada y la imagen segmentada para evaluar los resultados. En comparación con el modelo RGB, donde la segmentación requiere manejar la interacción compleja entre las componentes de color, el modelo HSV simplifica este proceso, ya que permite trabajar directamente sobre el atributo tonal (color), haciendo el procedimiento más intuitivo y menos propenso a errores por variaciones en brillo o saturación.

Solución

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import sys

```

```

4  from ip_functions import *
5
6  # Abrir imagen
7  RGB=plt.imread("cartagena.jpg")
8
9  #Realizar Slicing para reducir procesamiento
10 S=10
11 RGB=np.array(RGB[1::S,1::S,:])
12
13 """#Extraer Capas HSV"""
14
15 # RGB es la imagen de entrada
16 HSV=rgb2HSV(RGB)
17 h,s,v= imsplit(HSV)
18
19 """#Segmenta HSV"""
20 #Segmentar el color Amarillo
21 # Definir umbrales para H (tono)
22 UmbralInferiorH, UmbralSuperiorH = 0/360, 98/360
23 # Definir umbrales para S (saturación)
24 UmbralInferiorS, UmbralSuperiorS = 0.72, 1
25 # Definir umbrales para V (valor)
26 UmbralInferiorV, UmbralSuperiorV = 0.48, 0.76
27
28 # Crear máscaras
29 MascaraH = (h >= UmbralInferiorH) & (h <= UmbralSuperiorH)
30 MascaraS = (s >= UmbralInferiorS) & (s <= UmbralSuperiorS)
31 MascaraV = (v >= UmbralInferiorV) & (v <= UmbralSuperiorV)
32
33 # Graficar resultados
34 plt.figure(figsize=(15, 10))
35 plt.subplot(2, 3, 1)
36 plt.imshow(h, cmap='gray')
37 plt.title('Hue (Tono)')
38
39 plt.subplot(2, 3, 2)
40 plt.imshow(s, cmap='gray')
41 plt.title('Saturation (Saturación)')
42
43 plt.subplot(2, 3, 3)
44 plt.imshow(v, cmap='gray')
45 plt.title('Value (Valor)')
46
47 plt.subplot(2, 3, 4)
48 plt.imshow(MascaraH, cmap='gray')
49 plt.title('Máscara Hue')
50
51 plt.subplot(2, 3, 5)
52 plt.imshow(MascaraS, cmap='gray')
53 plt.title('Máscara Saturation')
54
55 plt.subplot(2, 3, 6)
56 plt.imshow(MascaraV, cmap='gray')
57 plt.title('Máscara Value')
58
59 plt.tight_layout()
60
61 # Combinar máscaras
62 MascaraFinal = MascaraH & MascaraS & MascaraV
63
64 """#Imagen Segmentada en HSV"""
65 # Graficar resultados
66 plt.figure(2, figsize=(20, 10))
67 plt.subplot(1, 3, 1)
68 plt.imshow(RGB)
69 plt.title('Imagen RGB Original')
70 plt.subplot(1, 3, 2)
71 plt.imshow(MascaraFinal, cmap='gray')
72 plt.title('Máscara Final')
73 plt.subplot(1, 3, 3)
74 plt.imshow(MascaraFinal[:, :, np.newaxis] * RGB)

```

```

75 plt.title('Imagen HSV Enmascarada')
76
77 plt.show()

```

La Fig.5.14 muestra la segmentación de la franja amarilla de la bandera de Colombia utilizando el modelo HSV. La parte superior muestra las representaciones de las componentes de tonalidad (H), saturación (S) y valor (V), permitiendo observar cómo el modelo HSV separa el color de sus características de brillo e intensidad. En el centro, se presentan las máscaras binarias generadas utilizando rangos definidos específicamente para el color amarillo en el espacio HSV, lo que facilita una segmentación más precisa e intuitiva. En la parte inferior, se observa la imagen original a color, seguida de la máscara generada y, finalmente, la región amarilla segmentada. Este enfoque destaca las ventajas del modelo HSV sobre el modelo RGB, especialmente al trabajar con colores específicos en presencia de variaciones de iluminación o saturación. Las funciones referidas para esta segmentación se encuentran explicadas en el capítulo 13.

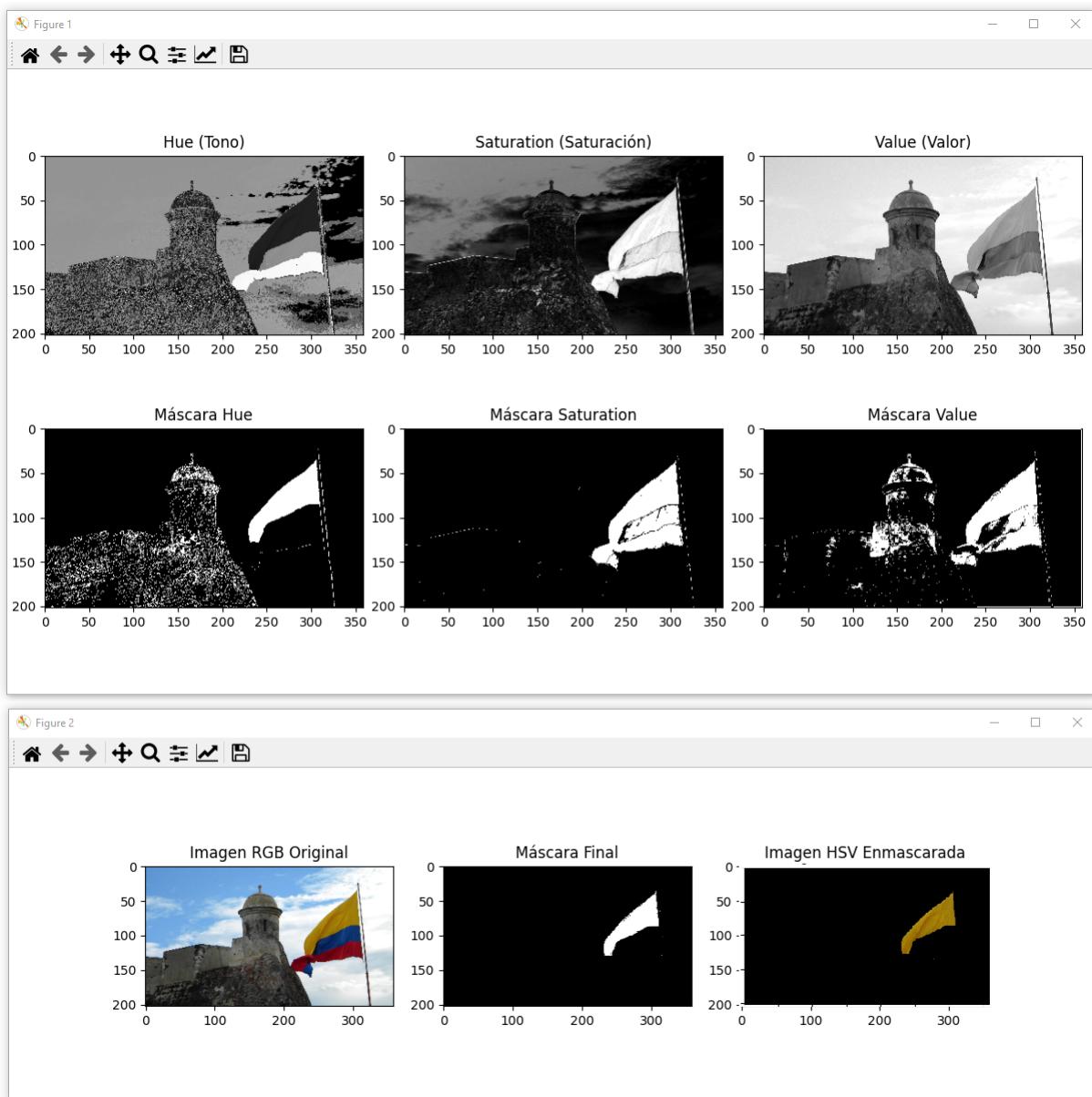


Figura 5.14: Segmentación de la bandera de Colombia en su franja amarilla usando el modelo HSV. En la parte inferior, se muestran la imagen a color, luego la máscara de segmentación y, finalmente, la región amarilla segmentada.

5.3.3. Ejemplo en Python: Segmentación de la bandera usando el modelo CieLab

Este código en Python realiza la segmentación del color amarillo en la bandera de la imagen 'cartagena.jpg' utilizando el modelo de color CIELAB, destacándose por su precisión y correspondencia con la percepción humana del color. El proceso comienza con la carga de la imagen y su conversión al espacio de color CIELAB, donde se trabajan las componentes L* (luminosidad), a* (verde a rojo) y b* (azul a amarillo). Se definen rangos específicos en las componentes a* y b*, que corresponden al color amarillo, mientras que L* puede utilizarse para ajustar el impacto de la iluminación, logrando una segmentación robusta frente a variaciones en las condiciones de luz. Utilizando estos rangos, se genera una máscara binaria que identifica las regiones de la imagen que cumplen con los criterios del color amarillo. Esta máscara se aplica posteriormente a la imagen original para extraer únicamente las áreas de interés. Finalmente, se presentan la imagen original, la máscara generada y la imagen segmentada para evaluar los resultados.

Solución

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import sys
4 from ip_functions import *
5
6 # Abrir imagen
7 RGB=plt.imread("cartagena.jpg")
8
9 #Realizar Slicing para reducir procesamiento
10 S=10
11 RGB=np.array(RGB[1::S,1::S,:])
12
13 """#Extraer Capas Cielab"""
14 # RGB es la imagen de entrada
15 LAB=rgb2lab(RGB)
16 # Separar los canales LAB
17 L, a, b= imsplit(LAB)
18
19 """#Segmenta LAB"""
20 # Segmentar el color Amarillo en CIELab
21 # Definir umbral para l (luminosidad)
22 UmbralInferiorL, UmbralSuperiorL = 37, 99
23 # Definir umbral para a (componente a)
24 UmbralInferiorA, UmbralSuperiorA = -10, 25
25 # Definir umbral para b (componente b)
26 UmbralInferiorB, UmbralSuperiorB = 40, 16
27
28
29 # Crear máscaras
30 MascaraL = (L >= UmbralInferiorL) & (L <= UmbralSuperiorL)
31 MascaraA = (a >= UmbralInferiorA) & (a <= UmbralSuperiorA)
32 MascaraB = (b >= UmbralInferiorB) & (b <= UmbralSuperiorB)
33
34 # Graficar resultados
35 plt.figure(figsize=(15, 10))
36 plt.subplot(2, 3, 1)
37 plt.imshow(L, cmap='gray')
38 plt.title('L (Luminosidad)')
39
40 plt.subplot(2, 3, 2)
41 plt.imshow(a, cmap='gray')
42 plt.title('a (Verde-Rojo)')
43
44 plt.subplot(2, 3, 3)
45 plt.imshow(b, cmap='gray')
46 plt.title('b (Azul-Amarillo)')
47
48 plt.subplot(2, 3, 4)
49 plt.imshow(MascaraL, cmap='gray')
50 plt.title('Máscara L')
51
52 plt.subplot(2, 3, 5)
53 plt.imshow(MascaraA, cmap='gray')
```

```

54 plt.title('Máscara a')
55
56 plt.subplot(2, 3, 6)
57 plt.imshow(MascaraB, cmap='gray')
58 plt.title('Máscara b')
59
60 plt.tight_layout()
61
62 # Combinar máscaras
63 MascaraFinal = MascaraL & MascaraA & MascaraB
64
65 """#Imagen Segmentada en HSV"""
66 # Graficar resultados
67 plt.figure(2, figsize=(20, 10))
68 plt.subplot(1, 3, 1)
69 plt.imshow(RGB)
70 plt.title('Imagen RGB Original')
71 plt.subplot(1, 3, 2)
72 plt.imshow(MascaraFinal, cmap='gray')
73 plt.title('Máscara Final')
74 plt.subplot(1, 3, 3)
75 plt.imshow(MascaraFinal[:, :, np.newaxis] * RGB)
76 plt.title('Imagen CieLAB Enmascarada')
77
78 plt.show()

```

La Fig.5.15 muestra la segmentación de la franja amarilla de la bandera de Colombia utilizando el modelo CIELAB. La parte superior muestra las representaciones de las componentes L* (luminosidad), a* (verde a rojo) y b* (azul a amarillo), evidenciando cómo este modelo separa la percepción del color de las condiciones de iluminación. En el centro, se presentan las máscaras binarias generadas utilizando rangos definidos específicamente en las componentes a* y b* para identificar el color amarillo, proporcionando una segmentación consistente. En la parte inferior, se observa la imagen original a color, seguida de la máscara generada y, finalmente, la región amarilla segmentada. Este enfoque resalta las ventajas del modelo CIELAB sobre los modelos RGB y HSV, ya que permite una segmentación más precisa al estar alineado con la percepción humana del color y ser menos sensible a las variaciones de iluminación. Las funciones utilizadas para esta segmentación están descritas en detalle en el capítulo 13.

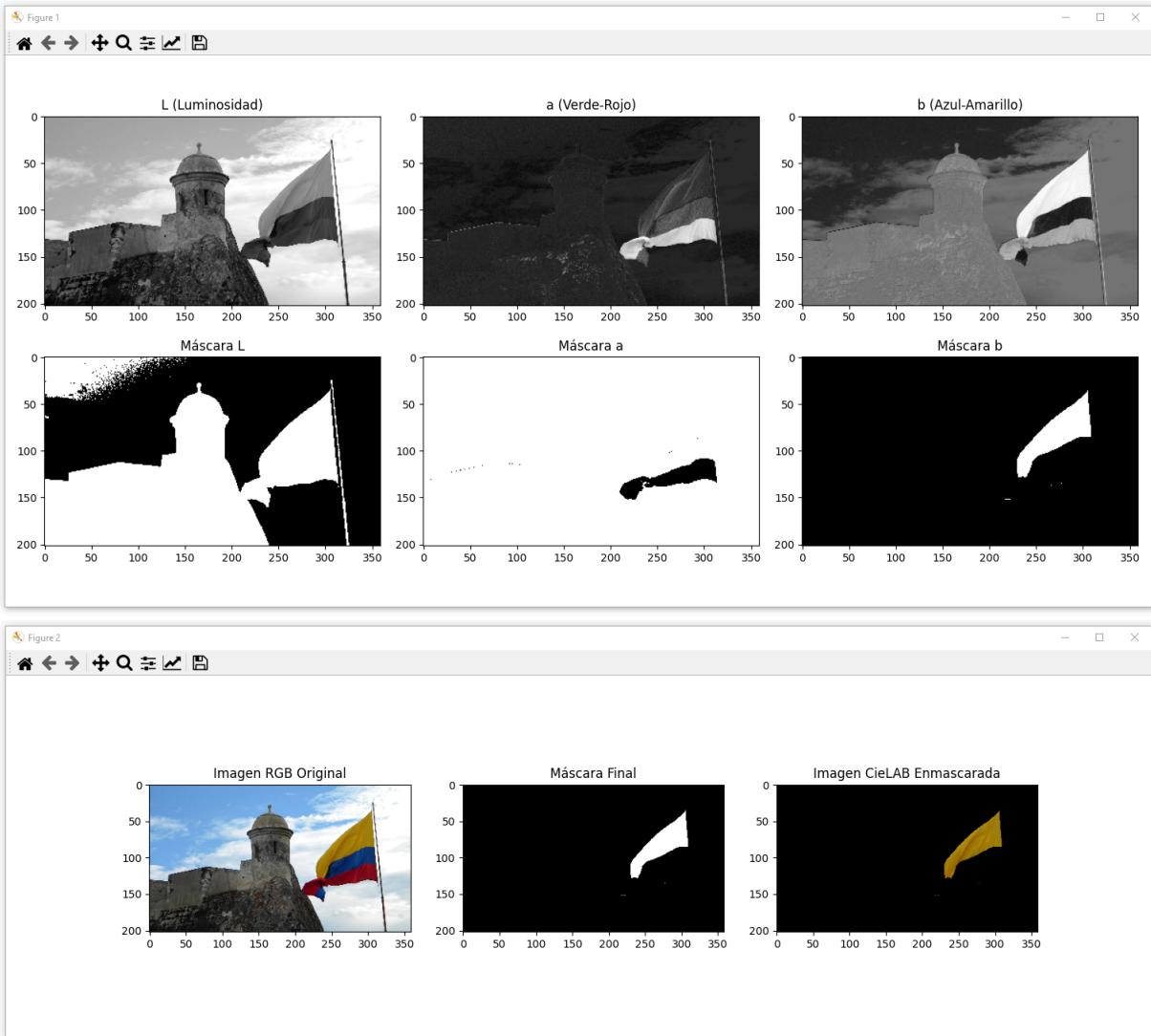


Figura 5.15: Segmentación de la bandera de Colombia en su franja amarilla usando el modelo CieLab. En la parte inferior, se muestran la imagen a color, luego la máscara de segmentación y, finalmente, la región amarilla segmentada.

5.4. Funciones

Las siguientes funciones tienen relación con el tema abordado en este capítulo:

1. `rgb2lab(I)`: convierte una imagen RGB al espacio de color $L^*a^*b^*$, aproximando la percepción visual humana. La imagen de entrada I debe tener tres canales y valores normalizados entre 0 y 1.
2. `lab2rgb(Lab)`: realiza la conversión inversa de `rgb2lab`, transformando una imagen en el espacio $L^*a^*b^*$ al espacio RGB. La imagen Lab debe tener tres canales (L , a , b) y los valores de salida estarán en el rango $[0, 1]$.
3. `xyz2lab(XYZ)`: convierte una imagen del espacio de color CIE 1931 XYZ al espacio $L^*a^*b^*$, usando la relación definida por la CIE para representar colores de forma perceptual. La imagen de entrada debe contener tres canales con valores normalizados.
4. `lab2xyz(Lab)`: transforma una imagen desde el espacio $L^*a^*b^*$ al espacio XYZ, revirtiendo la conversión perceptual al espacio de referencia CIE 1931.

5. `rgb2xyz(I)`: convierte una imagen RGB al espacio XYZ, que es un modelo de color lineal usado como base en muchas conversiones de color. La entrada `I` debe estar normalizada entre 0 y 1.
6. `xyz2rgb(XYZ)`: convierte una imagen en el espacio XYZ al espacio RGB, aplicando transformaciones para obtener una imagen adecuada para visualización en sRGB.
7. `hsv2rgb(HSV)`: transforma una imagen del espacio HSV (Matiz, Saturación, Valor) al espacio RGB, permitiendo representar colores definidos por sus características perceptuales. La imagen de entrada debe tener valores normalizados.
8. `rgb2hsv(I)`: convierte una imagen RGB al espacio HSV, separando la información de tono, saturación y brillo, lo cual es útil para análisis o segmentación basada en color. La imagen RGB debe tener valores entre 0 y 1.
9. `imshow(I, cmap=None, vmin=None, vmax=None)`: muestra la imagen `I` en una ventana gráfica. Opcionalmente, se puede indicar un mapa de colores (`cmap`) y establecer los valores mínimos y máximos de intensidad a mostrar mediante `vmin` y `vmax`.
10. `imread(nombre_archivo)`: lee una imagen desde un archivo y la carga como un arreglo numpy. El nombre del archivo debe incluir su ruta o extensión (por ejemplo, 'imagen.jpg'). La imagen resultante puede ser en escala de grises o en color, según el archivo.

5.5. Preguntas

5.5.1. Preguntas directas sobre el texto

1. ¿Cuál es la principal razón evolutiva propuesta para el origen de la visión del color en los animales, según el texto?
2. En la teoría simplificada de la visión del color mencionada, ¿cómo se produce el color amarillo a partir de los estímulos de los conos en la retina?
3. ¿Cuál es la diferencia principal entre el observador estándar de 1931 y el de 1964 en el modelo de color CIE XYZ?
4. Segundo el texto, ¿por qué se necesitó implementar el modelo teórico CIE XYZ a pesar de la existencia del modelo CIE RGB?
5. Qué ventajas hay entre entre el modelo HSV y el RGB?

5.5.2. Preguntas de análisis y compresión con base en el texto

1. ¿Por qué el modelo HSV es más intuitivo que el modelo RGB para la selección de colores en dispositivos como las lámparas RGB?
2. Dado que cada capa de color (rojo, verde y azul) en una imagen se visualiza como una sola tonalidad, ¿cómo se representa el color amarillo en términos de la saturación y el brillo, y cuál es la contribución de cada capa de la imagen (rojo, verde y azul) para producir ese color?
3. En el ejemplo de segmentación de color amarillo de la bandera colombiana, ¿qué modelo de color (RGB, HSV o CIELAB) parece ser el más intuitivo y fácil de usar para esta tarea? Explique su respuesta.
4. ¿Qué utilidad tiene el modelo CMYK? de ejemplo de uso.

5.5.3. Preguntas de cálculos

1. Dada la matriz de colores RGB a HSV , presente los resultados en una tabla con las columnas: Pixel, C_{\max} , C_{\min} , δ , H , S y V . Realice todos los cálculos a mano, mostrando cada paso del proceso.

$$R = \begin{bmatrix} 0 & 255 \\ 255 & 0 \end{bmatrix} \quad G = \begin{bmatrix} 255 & 0 \\ 255 & 255 \end{bmatrix} \quad B = \begin{bmatrix} 255 & 255 \\ 0 & 0 \end{bmatrix}$$

Pixel	r	g	b	C_{\max}	C_{\min}	δ	H	H normalizado	S	V
(1, 1)										
(1, 2)										
(2, 1)										
(2, 2)										

Cuadro 5.9: Cálculos completos de C_{\max} , C_{\min} , δ , H , S y V .

2. El resultado de la conversión obtenido en HSV del punto 1, llevarlo nuevamente a RGB mostrado en una tabla las columnas Pixel, C , X , m , r , g y b . Realice todos los cálculos a mano, mostrando cada paso del proceso.

Pixel	H	S	V	C	X	m	r	g	b	R	G	B
(1, 1)												
(1, 2)												
(2, 1)												
(2, 2)												

Cuadro 5.10: Cálculos numéricos de C , X , m , r , g , b , R , G y B para cada pixel.

3. Dada la matriz de colores RGB a XYZ , presente los resultados en una tabla con las columnas Pixel, r , g , b , r_l , g_l , b_l , X , Y , Z , X_n , Y_n , Z_n . Realice todos los cálculos a mano, mostrando cada paso del proceso.

$$R = \begin{bmatrix} 0 & 255 \\ 255 & 0 \end{bmatrix} \quad G = \begin{bmatrix} 255 & 0 \\ 255 & 255 \end{bmatrix} \quad B = \begin{bmatrix} 255 & 255 \\ 0 & 0 \end{bmatrix}$$

Pixel	r	g	b	r_l	g_l	b_l	X	Y	Z	X_n	Y_n	Z_n
(1,1)												
(1,2)												
(2,1)												
(2,2)												

Cuadro 5.11: Cálculos numéricos de r_l , g_l , b_l , X , Y , Z , X_n , Y_n , Z_n , de cada pixel dela conversión de RGB a XYZ.

4. El resultado de la conversión obtenido en XYZ del punto anterior, llevarlo nuevamente a RGB mostrado en una tabla las columnas Pixel, X , Y , Z , r , g , b , r_l , g_l , b_l , R , G , B . Realice todos los cálculos a mano, mostrando cada paso del proceso.

Pixel	X	Y	Z	r	g	b	r_l	g_l	b_l	R	G	B
(1,1)												
(1,2)												
(2,1)												
(2,2)												

Cuadro 5.12: Cálculos numéricos de $X, Y, Z, r, g, b, r_l, g_l, b_l, R, G, B$, de cada pixel dela conversión de XYZ a RGB.

5. El resultado de la conversión obtenida en XYZ del punto 3, llevarlo a *CieLab*. los cálculos numéricos de $X, Y, Z, x_r, y_r, z_r, f_x, f_y, f_z, L, a, b$ de cada pixel (i, j) . Realice todos los cálculos a mano, mostrando cada paso del proceso.

Pixel	X	Y	Z	x_r	y_r	z_r	f_x	f_y	f_z	L	a	b
(1,1)												
(1,2)												
(2,1)												
(2,2)												

Cuadro 5.13: Cálculos numéricos de $X, Y, Z, x_r, y_r, z_r, f_x, f_y, f_z, L, a, b$ de cada pixel de la conversión XYZ a Lab .

6. El resultado de la conversión obtenido en *Cielab* del punto 5, llevarlo nuevamente a XYZ mostrado en una tabla las columnas Pixel, $L, a, b, f_y, f_x, f_z, x_r, y_r, z_r, X, Y, Z$. Realice todos los cálculos a mano, mostrando cada paso del proceso.

Pixel	L	a	b	f_y	f_x	f_z	x_r	y_r	z_r	X	Y	Z
(1,1)												
(1,2)												
(2,1)												
(2,2)												

Cuadro 5.14: Cálculos numéricos de $L, a, b, f_y, f_x, f_z, x_r, y_r, z_r, X, Y, Z$, de cada pixel de la conversión de Lab a XYZ .

Capítulo 6

Conversión a escala de grises

Como ya se ha discutido, una imagen RGB cada píxel se especifica mediante tres valores, uno para cada componente del color rojo, azul y verde. En ella la matriz resultante es tridimensional con valores enteros entre [0, 255] o double entre [0, 1]. La formación matricial que usa sensores u otros equipos de adquisición que solo involucran intensidad ,sin tener en cuenta todas las fuentes espectrales, se denomina imagen en escala de grises. Estas, por lo tanto, solo requieren de una matriz (2D) para su representación [7,56].

La conversión de una imagen a color a escala de grises es una técnica básica en el procesamiento de imágenes, ya que simplifica el análisis visual y computacional. Aunque las imágenes a color contienen una gran cantidad de información distribuida en los canales rojo, verde y azul, en muchas aplicaciones no es indispensable. Por ejemplo, la escala de grises es suficiente para tareas como la detección de bordes, la segmentación o el reconocimiento de patrones, ya que conserva la información clave de la intensidad luminosa. Por otro lado, al trabajar con matrices bidimensionales en lugar de tridimensionales, se reduce la complejidad computacional y se optimiza el uso de recursos, lo que resulta útil en aplicaciones donde la eficiencia es importante. De igual forma, las imágenes en escala de grises requieren también menos espacio de almacenamiento y son compatibles con algoritmos tradicionales diseñados para este formato, como los métodos de detección de bordes (Canny, Sobel) y otras transformaciones matemáticas .

6.1. El problema

Llevar una imagen a color en RGB (3D) a su equivalente en escala de grises (2D), parece un problema sencillo de resolver. En principio, podría considerarse que tomar cualquiera de las capas de color representaría una versión de la imagen de la bandera en la Fig.6.1. No obstante, debe recordarse que cada capa solo contiene información espectral de la tonalidad que lleve el color. Así las cosas, la capa roja muestra claros los rojos y oscuros el color azul. Como puede notarse, el color amarillo de la bandera también se muestra algo blanco en la capa roja, ya que este es la mezcla de rojo y verde. Por lo anterior, la capa del color verde entrega también blanco el color amarillo y oscuro los colores rojo y azul. Finalmente, en la capa azul, este color se ve blanco para dicha franja de la bandera y los restantes colores lucen negros. Nótese que en esta última capa, el cielo se ve más claro por el dominio de dicho color y las nubes se aprecian blancas en todas, ya que este color resulta de la combinación aditiva de todos los colores base [56] .

6.2. Técnica del promedio

La conversión a escala de grises mediante el método del promedio es una de las formas más intuitivas, pero presenta limitaciones en términos de mostrar fielmente la percepción humana de la imagen. Esto se debe a que los colores base tienen diferentes longitudes de onda y contribuyen de manera distinta a la percepción visual debido a la variación en la cantidad de conos presentes en la retina. Aunque este método es simple y eficiente computacionalmente al calcular el promedio de los valores de los canales rojo, verde y azul de cada píxel, conservando así cierta información del color y los detalles originales de la imagen, no toma en cuenta la sensibilidad relativa del ojo humano a diferentes colores.

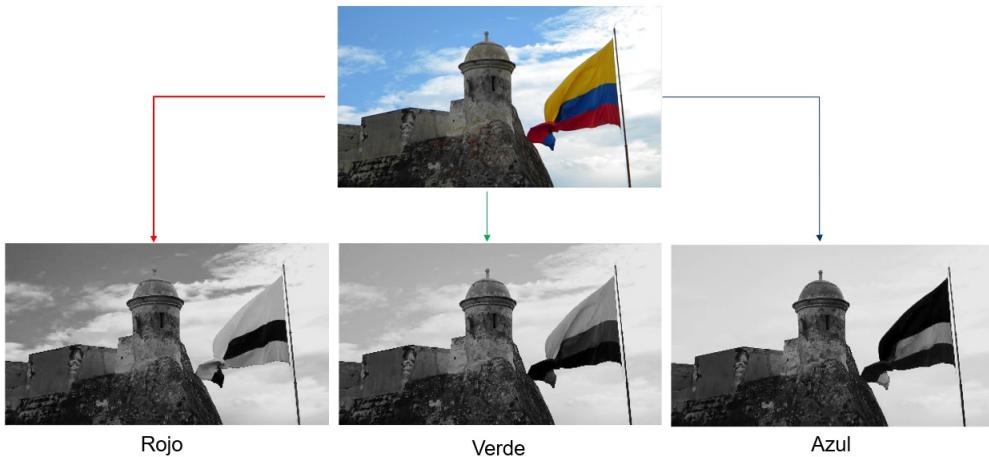


Figura 6.1: Descomposición de las campas del modelo RGB de la imagen de la bandera.

Por lo tanto, puede resultar en una representación menos exacta en términos de la luminosidad y el contraste en la imagen convertida a escala de grises [7,27,56]. Véase la Figura 6.2.

$$Y = w_R \cdot R + w_G \cdot G + w_B \cdot B \quad \text{donde} \quad w_R = \frac{1}{3}, \quad w_G = \frac{1}{3}, \quad w_B = \frac{1}{3}$$

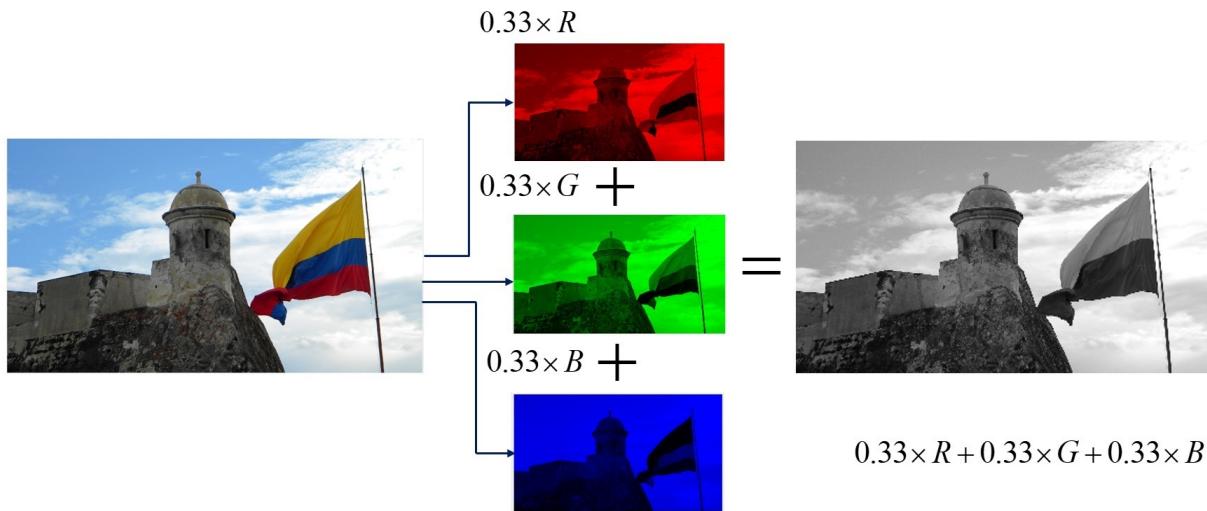


Figura 6.2: Imagen a color y su respectiva representación en escala de grises mediante el promedio simple.

6.3. Técnica del Midgray

La escala de grises mediante Midgray se define como una combinación de la media de las componentes de color máxima y mínima. La conversión de imágenes a escala de grises corresponde a asignar un valor constante de gris a cada píxel, generalmente un tono intermedio de gris. Esto proporciona una solución simple y computacionalmente eficiente, manteniendo un nivel básico de contraste en la imagen resultante. No obstante, esta simplicidad conlleva una pérdida considerable de información y detalle de color, ya que no se considera la variación en la intensidad de los píxeles. Además, al no adaptarse a las características específicas de la imagen, como su contenido y distribución de colores, esta puede no ser adecuada para todas las imágenes y situaciones, lo que limita su utilidad en situaciones que demanden una representación más rigurosa de la imagen en escala de grises [7, 27, 56]. Véase la Figura 6.3.

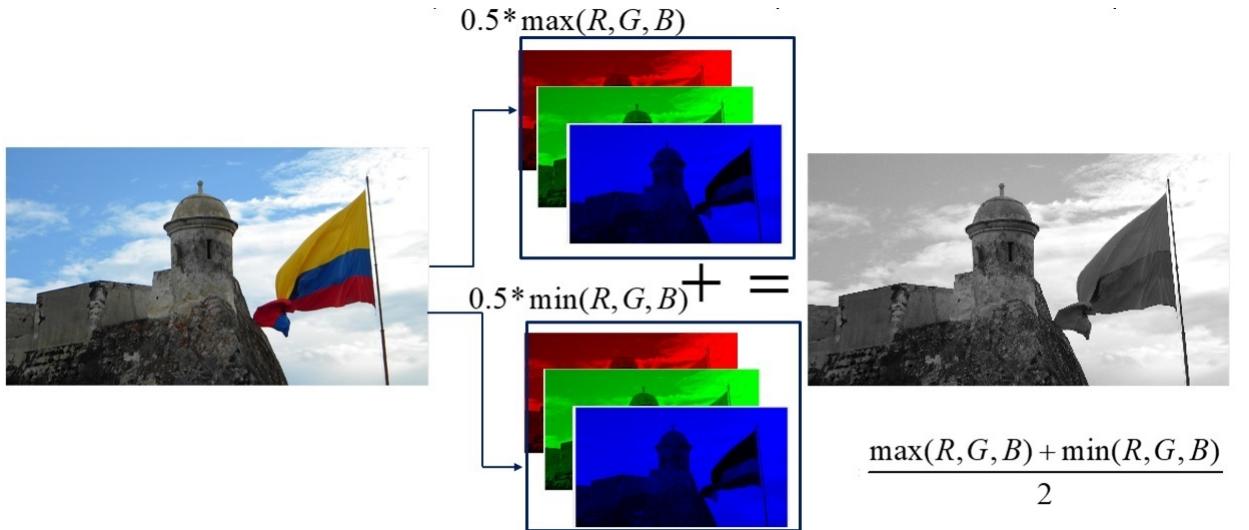


Figura 6.3: Imagen a color y su respectiva representación en escala de grises usando la luminancia.

6.4. Técnica de la luminancia

La conversión de una imagen a color RGB a una representación en escala de grises se logra mediante el cálculo del valor de la *luminancia* Y para cada pixel RGB. En su forma más simple, Y se calcula como el promedio ponderado de las tres componentes de color R, G y B. Este modelo considera de mayor peso el color verde, el cual corresponde al mayor número de conos de la retina humana. Lo anterior hace la imagen más brillante.

$$Y = lum(R, G, B) = w_R \cdot R + w_G \cdot G + w_B \cdot B \quad (6.1)$$

Las ponderaciones utilizadas con mayor frecuencia fueron desarrolladas originalmente para la codificación de señales analógicas de televisión a color según la recomendación NTSC 601. Esta norma técnica se utiliza para la codificación de video analógico en televisión. No se relaciona directamente con la conversión a escala de grises de imágenes digitales, pero puede ser útil para entender cómo se maneja la información de color en diferentes sistemas de video. La conversión a escala de grises de una imagen digital implica la eliminación de toda la información de color y la conservación de solo la información de luminancia. Esto se puede hacer mediante la supresión de los componentes de color de una imagen RGB o mediante la conversión de una imagen en formato YUV a escala de grises utilizando solo el canal Y (que representa la luminancia). En el formato YUV, el canal Y contiene información de luminancia, mientras que los canales U y V contienen información de color. La norma NTCS 601 especifica la forma en que se codifica la información de color en el formato YUV para la televisión analógica, y se basa en una matriz de conversión de RGB a YUV que se utiliza para convertir las señales de video analógicas a formato digital [7, 27, 56].

$$w_R = 0.299 \quad w_G = 0.587 \quad w_B = 0.114$$

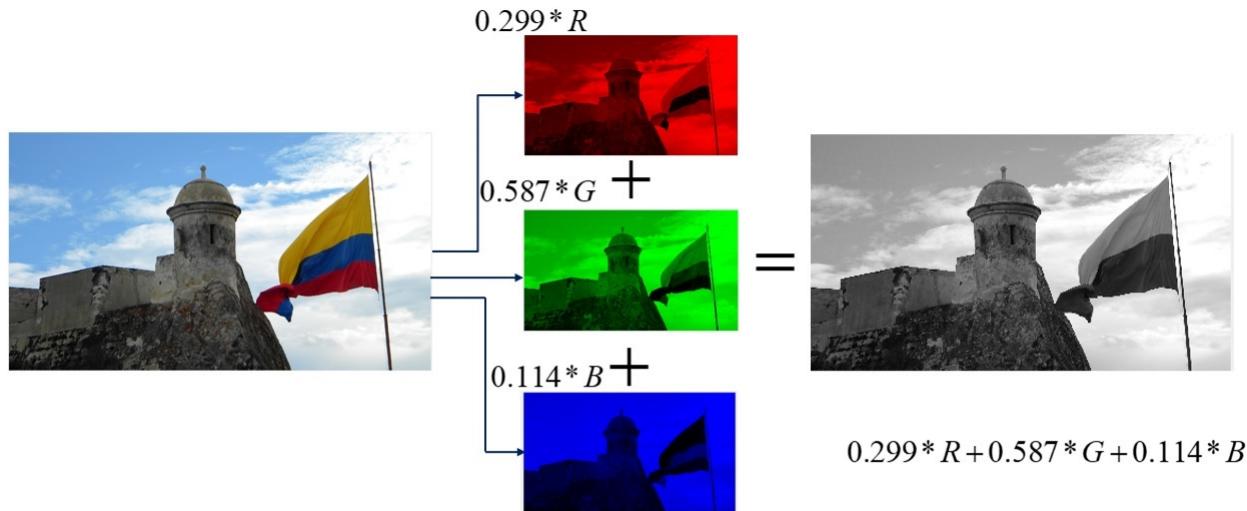


Figura 6.4: Imagen a color y su respectiva representación en escala de grises usando la luminancia. . La recomendación NTSC 601 es la más utilizada.

Por otro lado, la Rec. 709 es un estándar internacional que establece las especificaciones para la producción, transmisión y visualización de imágenes de televisión en color de alta definición (HD). También se conoce como ITU-R BT.709 o simplemente BT.709; fue desarrollada por la Unión Internacional de Telecomunicaciones (ITU) en 1990. La conversión en escala de grises con base en dicha recomendación especifica también los valores de ponderación para las componentes rojo, verde y azul establecidas experimentalmente para producir una imagen en escala de grises que parece más natural al ojo humano. Nótese que los valores de la Rec 709 y NTSC 601 son diferentes.

$$w_R = 0.2226, \quad w_G = 0.7152, \quad w_B = 0.0722$$

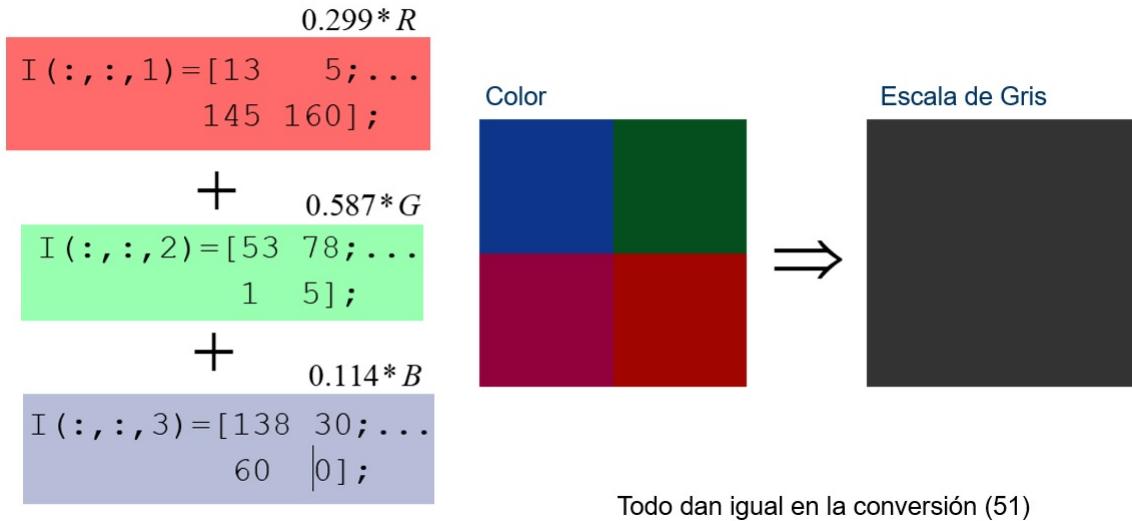
6.5. Conversión de escala de grises a color

De entrada debe considerarse que no existe un método inverso para lograr llevar una imagen en escala de grises a sus colores originales. si se revisa el modelo más usando que corresponde a la luminancia por NTSC 601, este realiza un promedio ponderado que puede ser el resultado de la combinación lineal de varios valores para un mismo pixel. En la Figura 6.5 se ilustra el problema de la siguiente manera. Para todos los colores mostrados visualmente diferentes respecto al color, producen la misma equivalencia en escala de gris. Es decir, existe un amplio rango de valores de rojo, verde y azul que ponderados generan el mismo valor de salida.

No obstante, una imagen en escala de grises se puede convertir en una versión coloreada mediante el procesamiento de falso color, el cual se usa a menudo para la representación de datos como gradientes de temperatura, información meteorológica, y rayos x entre otras aplicaciones. Para producir una imagen en color similar a la imagen en color original, es necesario conocer más información sobre la imagen original. En el caso de piel humana, se puede estimar una aproximación a su tonalidad, pero es solo una representación. Con base en este principio, se han desarrollado algoritmos que reconocen algunos objetos y los colorean de acuerdo a su naturaleza tales como el firmamento, el mar, y los árboles [1, 26].

6.5.1. Ejemplo en Python: Segmentación por Color y Grises

En este ejemplo se realiza la segmentación de una imagen en el espacio de color HSV con el objetivo de identificar y resaltar un automóvil de color amarillo, dejando el resto de la imagen en escala de grises. Inicialmente, se convierte del espacio de color RGB a HSV para facilitar la detección del color amarillo. Se establecen umbrales específicos para los canales H, S y V con el fin de generar una máscara que identifique las regiones de interés. Posteriormente, se aplica la máscara a la imagen original para extraer solo las áreas correspondientes al color amarillo, mientras que el resto de la imagen se convierte a escala de grises para resaltar la segmentación. Finalmente, se visualizan los



Todo dan igual en la conversión (51)

Figura 6.5: Conversión directa de gris a color con base en NTSC 601.

resultados, incluyendo la imagen original, la máscara aplicada y la imagen segmentada, permitiendo resaltar objetos de color amarillo en la imagen utilizando técnicas de procesamiento de imágenes basadas en el espacio de color HSV.

Solución

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import sys
4 from ip_functions import *
5
6
7 # Abrir imagen
8 RGB=plt.imread("carro.JPG")
9
10 #Realizar Slicing para reducir procesamiento
11 S=10
12 RGB=np.array(RGB[1::S,1::S,:])
13
14 #Extraer Capas HSV"""
15
16 # RGB es la imagen de entrada
17 HSV=rgb2HSV(RGB)
18 h,s,v=imsplit(HSV)
19
20 #Segmenta HSV"""
21
22 # imagen de entrada en formato RGB
23 HSV=rgb2HSV(RGB)
24
25 # Separar los canales HSV
26 h, s, v = imsplit(HSV)
27
28 #Segmentar ewl color Amarillo
29
30 # Definir umbrales para H (tono)
31 UmbralInferiorH, UmbralSuperiorH =20/360, 60/360
32 # Definir umbrales para S (saturación)
33 UmbralInferiorS, UmbralSuperiorS = 0.5, 1
34 # Definir umbrales para V (valor)
35 UmbralInferiorV, UmbralSuperiorV = 0.1, 1
36
37 # Crear máscaras
38 MascaraH = (h >= UmbralInferiorH) & (h <= UmbralSuperiorH)

```

```

39 MascaraS = (s >= UmbralInferiorS) & (s <= UmbralSuperiorS)
40 MascaraV = (v >= UmbralInferiorV) & (v <= UmbralSuperiorV)
41
42 # Graficar resultados
43 plt.figure(figsize=(15, 10))
44
45 plt.subplot(2, 3, 1)
46 plt.imshow(h, cmap='gray')
47 plt.title('Hue (Tono)')
48
49 plt.subplot(2, 3, 2)
50 plt.imshow(s, cmap='gray')
51 plt.title('Saturation (Saturación)')
52
53 plt.subplot(2, 3, 3)
54 plt.imshow(v, cmap='gray')
55 plt.title('Value (Valor)')
56
57 plt.subplot(2, 3, 4)
58 plt.imshow(MascaraH, cmap='gray')
59 plt.title('Máscara Hue')
60
61 plt.subplot(2, 3, 5)
62 plt.imshow(MascaraS, cmap='gray')
63 plt.title('Máscara Saturation')
64
65 plt.subplot(2, 3, 6)
66 plt.imshow(MascaraV, cmap='gray')
67 plt.title('Máscara Value')
68
69 plt.tight_layout()
70
71 #Imagen Color Segmentada en HSV
72
73 # Graficar resultados
74 # Combinar máscaras
75 MascaraColor = MascaraH & MascaraS & MascaraV
76 MascaraColor = MascaraColor[:, :, np.newaxis]
77 ColorSegmentado=MascaraColor* RGB
78 plt.figure(2, figsize=(20, 10))
79 plt.subplot(1, 3, 1)
80 plt.imshow(RGB)
81 plt.title('Imagen RGB Original')
82 plt.subplot(1, 3, 2)
83 plt.imshow(MascaraColor, cmap='gray')
84 plt.title('Máscara Final')
85 plt.subplot(1, 3, 3)
86 plt.imshow(ColorSegmentado)
87 plt.title('Imagen HSV Enmascarada')
88
89 #Imagen e Escala de Grises Segmentada
90 Gris=rgb2gray(RGB)
91 Gris=np.stack((Gris,), axis=-1)
92 MascaraGris=np.logical_not(MascaraColor)
93 GrisSegmentado=MascaraGris*Gris
94
95 plt.figure(3, figsize=(20, 10))
96 plt.subplot(1, 3, 1)
97 plt.imshow(Gris)
98 plt.title('Imagen Gris Original')
99 plt.subplot(1, 3, 2)
100 plt.imshow(MascaraGris, cmap='gray')
101 plt.title('Máscara GRis')
102 plt.subplot(1, 3, 3)
103 plt.imshow(GrisSegmentado)
104 plt.title('Imagen HSV Enmascarada')
105
106 #Imagen en escala de Grises con Color Segmentado
107
108 #Salida segmentada
109 Salida=GrisSegmentado+ColorSegmentado

```

```

110 plt.figure(4, figsize=(20, 10))
111 plt.subplot(1, 2, 1)
112 plt.imshow(RGB)
113 plt.title('Imagen a Color')
114 plt.subplot(1, 2, 2)
115 plt.imshow(Salida)
116 plt.title('Imagen Gris Segmentada')
117
118 plt.show()

```

En Fig.6.6 se precisa la segmentación del color amarillo para el carro utilizando el modelo HSV. La parte superior muestra las representaciones de las componentes de tonalidad (H), saturación (S) y valor (V), permitiendo observar cómo el modelo HSV separa el color de sus características de brillo e intensidad. En el centro, se presentan las máscaras binarias generadas utilizando rangos definidos específicamente para el color amarillo en el espacio HSV, lo que facilita una segmentación más precisa e intuitiva. Este enfoque destaca las ventajas del modelo HSV sobre el modelo RGB, especialmente al trabajar con colores específicos en presencia de variaciones de iluminación o saturación. Las funciones referidas para esta segmentación se encuentran explicadas en el capítulo 13.

6.6. Funciones

Las siguientes funciones tienen relación con el tema abordado en este capítulo:

1. `rgb2gray(I)`: convierte una imagen RGB a escala de grises aplicando una combinación ponderada de los canales rojo, verde y azul, de modo que se aproxime la percepción humana de la luminosidad. La imagen de entrada `I` debe tener tres canales con valores generalmente normalizados entre 0 y 1. Esta función devuelve una matriz bidimensional de intensidades en escala de grises.
2. `mat2gray(A)`: normaliza los valores de una matriz `A` al rango [0, 1] utilizando los valores mínimo y máximo de la matriz como referencia. Puede recibir como parámetros opcionales los límites inferior y superior deseados (`low`, `high`), lo cual permite escalar los datos dentro de un rango específico. Es útil para visualizar datos arbitrarios como imágenes o ajustar el contraste en procesos de previsualización.
3. `ind2gray(X, map)`: convierte una imagen indexada `X` y su correspondiente mapa de colores `map` en una imagen en escala de grises. La conversión se basa en calcular la luminosidad percibida de cada color del mapa y asignarla al píxel correspondiente en la imagen resultante. La matriz `X` contiene índices enteros y `map` es un arreglo de colores RGB.

6.7. Preguntas

6.7.1. Preguntas de Hechos Indicados en el Texto

1. ¿Por qué las imágenes en escala de grises requieren solo una matriz bidimensional para su representación?
2. ¿Cuáles son las técnicas de conversión a escala de grises enunciadas en el documento?
3. ¿Qué problema se plantea al intentar convertir una imagen RGB a escala de grises?
4. ¿Cuál de las técnicas le da más peso o ponderación a uno de los colores RGB, y por qué?
5. Entre NTSC 601 y Rec. 709, ¿cuál es más antigua y cuál es su uso principal?

6.7.2. Preguntas de Comprensión con Base en el Texto

1. ¿Por qué el verde tiene mayor peso en las técnicas de conversión con base en la luminancia?
2. ¿Es posible convertir imágenes en escala de grises de vuelta a color a partir de solo sus valores en gris?



Figura 6.6: Proceso de segmentación por color de la imagen de un auto amarillo usando el modelo del color HSV y la escala de grises.

3. ¿Cómo afectan las técnicas de conversión a la percepción visual en aplicaciones prácticas?
4. ¿Qué desafíos presenta la conversión a escala de grises en términos de la preservación de detalles?
5. ¿Cuál es la relación entre los estándares NTSC 601 y Rec. 709 y la evolución de la tecnología?

6.7.3. Ejercicios Numéricos con Base en el Texto

1. Dada una imagen con píxeles RGB $(R, G, B) = (120, 200, 80)$, calcule el valor en escala de grises usando las técnicas del promedio y Midgray.
2. Para un píxel RGB con valores $(80, 160, 100)$, calcule el valor en escala de grises usando NTSC 601 y Rec. 709, y discuta los resultados.
3. Dadas la intensidades $R = 10$ como valor mínimo, $Y_{\text{Midgray}} = 45$ y $Y_{\text{Promedio}} = 40$, determine los valores faltantes de B y G , sabiendo que esta última no es el máximo.
4. Dado un valor en escala de grises $Y = 128$ y los coeficientes de NTSC 601 y Rec. 709, determine el valor de R si $G = 100$ y $B = 50$ en cada caso. Comente si la solución es única.
5. Dado un píxel con luminancias conocidas $Y_{601} = 150$ (según NTSC 601) y $Y_{709} = 145$ (según Rec. 709), determine los valores de R y G si el valor de $B = 80$. Asegúrese de que los valores calculados para R y G estén en el rango entero entre $[0, 255]$.

Capítulo 7

Binarización

La binarización es un paso común en muchos algoritmos de análisis de imagen, como la detección de bordes, la segmentación de objetos y el reconocimiento de patrones. Al lograrlo la imagen, se puede preparar para su posterior procesamiento y análisis.

La binarización y la umbralización son técnicas que se utilizan para convertir una imagen en escala de grises en su representación binaria. Es de destacar que ambas técnicas no son exactamente iguales. La binarización lleva la imagen en escala de grises a su representación binaria con solo dos valores de intensidad de píxeles, es decir, valores de 0 y 1. En este proceso, se selecciona un valor de umbral fijo, y los píxeles con intensidad por debajo de ese valor se establecen en 0 (negro) y los píxeles con intensidad por encima del valor se establecen en 1 (blanco). Por otro lado, la umbralización permite segmentar la imagen en diferentes regiones según el valor de intensidad de los píxeles de la vecindad. En esta situación no hay un único umbral para toda la imagen.

La binarización de las imágenes es importante por algunas de estas razones:

Simplificación de la información: convierte una imagen en escala de grises en una imagen de solo dos valores de blanco y negro (unos y ceros). Esto reduce la complejidad de la información y permite que sea procesada de manera más eficiente.

Segmentación de objetos: los objetos en la imagen se vuelven más distinguibles y separables, lográndose una mejor segmentación y análisis de los elementos individuales.

Eliminación de ruido: ayuda a atenuar los elementos indeseados como sobras u otros artefactos, especialmente cuando se aplica un umbral adaptativo que ajusta para diferentes áreas.

La segmentación es un proceso fundamental en muchas aplicaciones de reconocimiento de patrones en las imágenes o los videos. A menudo la binarización se usa para dividir una escena en regiones separadas, que idealmente corresponden a diferentes objetos.

La binarización de imágenes es uno de los pasos más relevantes del preprocesamiento. Esta conduce a una disminución significativa de la información no relevante, lo que conlleva a una mejora en el rendimiento de las aplicaciones que no requieren análisis de textura o color de entrada. Algunas de estos usos se encuentran en la robótica móvil donde, en muchos casos, solo se requieren elementos básicos de la trayectoria para seguir una línea o en las aplicaciones del reconocimiento de caracteres, entre otras.

La binarización de una imagen es un tipo especial de cuantificación que separa el valor del pixel en dos clases, dependiendo de un valor de umbral dado T que usualmente es constante. La función de umbral $T(I)$ asigna todos

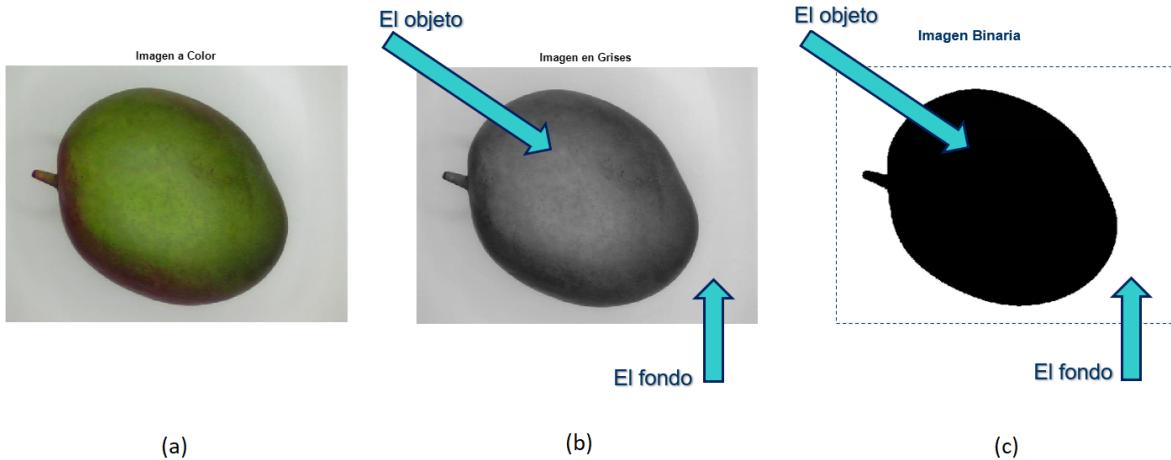


Figura 7.2

los pixeles a uno de dos valores de intensidad fijo a_0 o a_1 ; es decir:

$$T(I) = \begin{cases} a_0 = 0 & \text{para } I < T \\ a_1 = 1 & \text{para } I \geq T \end{cases} \quad (7.1)$$

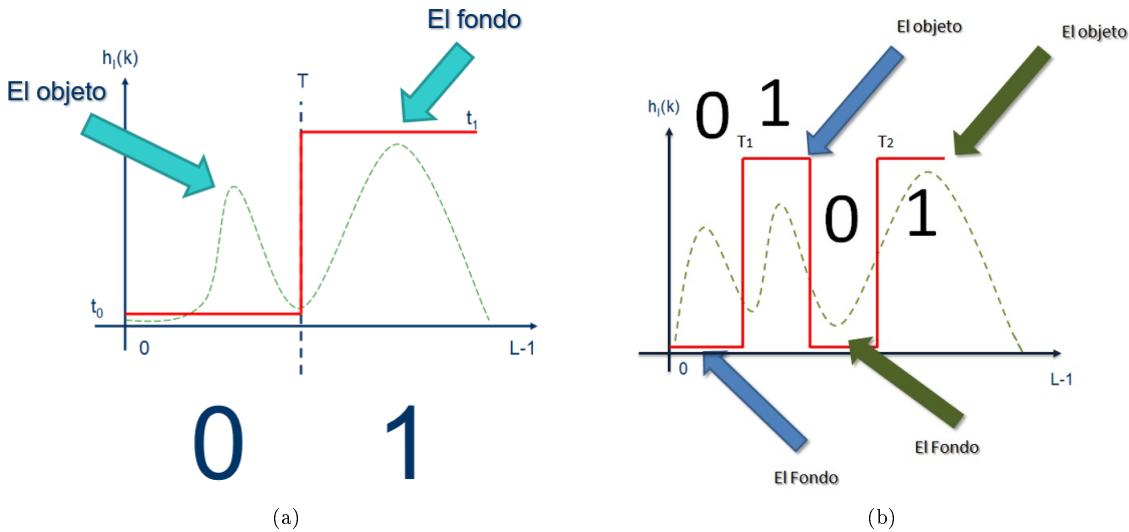


Figura 7.1: Efecto del umbral sobre el histograma. a) Binarización y b) Umbralización.

Para lograr la binarización de una imagen es importante definir cuál es el objeto de interés y cuál es el fondo. De manera iterativa, se puede procurar encontrar un umbral T el cual defina las regiones, pero este proceso no es siempre posible aplicarlo cuando el entorno no es controlado.

En la Fig. 7.3 se muestran las diferentes capas de modelo del color RGB para un mango. Allí se aprecia que, si bien todos son bimodales, la capa azul es la que mejor separa el objeto del fondo de interés tanto en la imagen como en su histograma bimodal. Allí es posible, dado que no los mangos no tienen componente de color azul, por lo que su

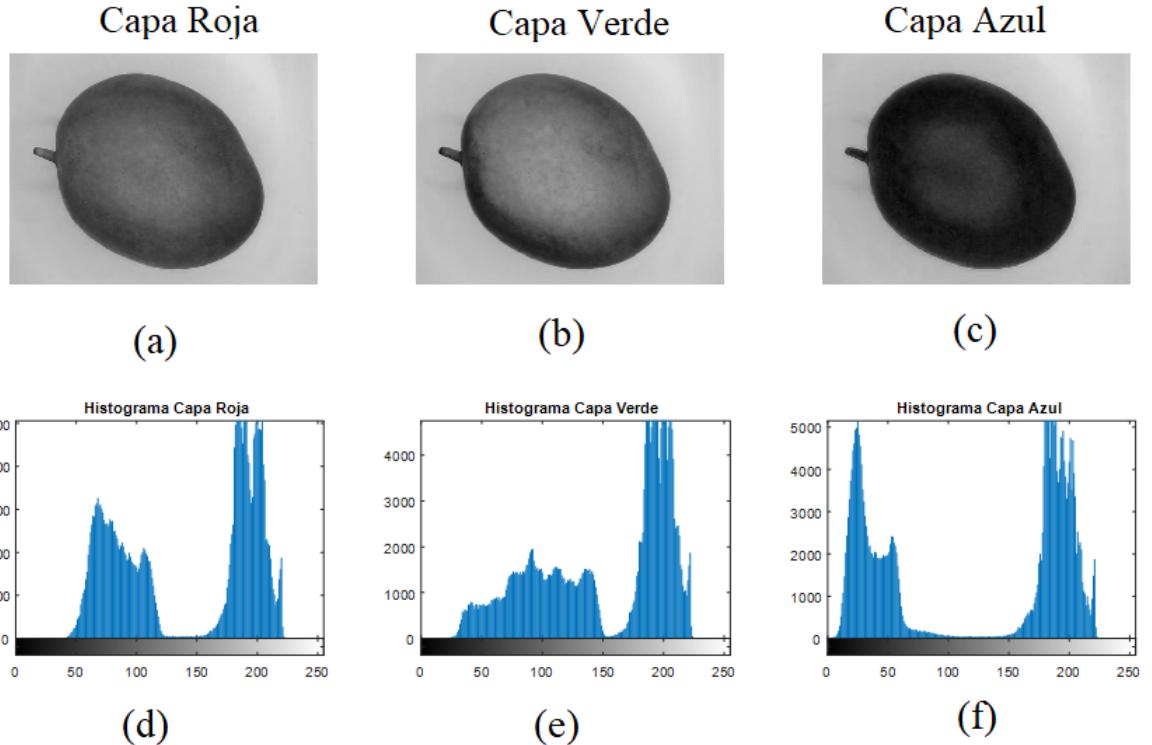
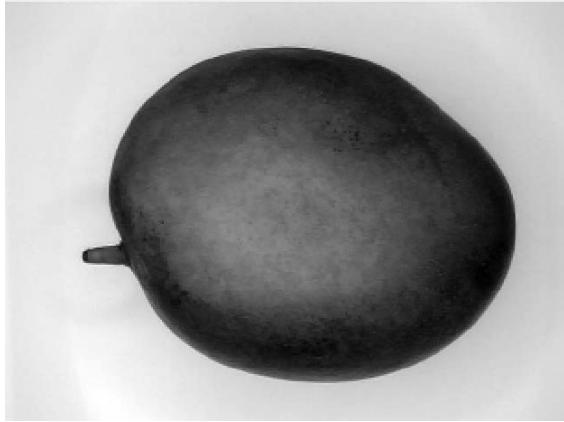


Figura 7.3: Histograma para cada una de las capas de color RGB.

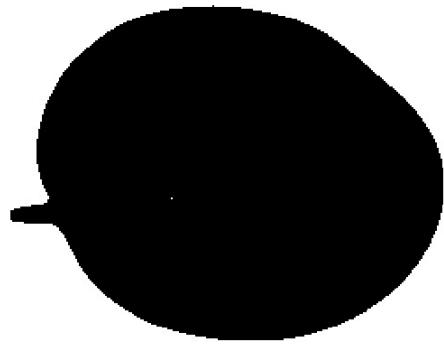
representación es más oscura. Por el contrario, el rojo y el verde son colores predominantes en muchas frutas que se pueden confundir con el blanco del fondo en su representación en escala de gris por tonalidad. Esta técnica puede fracasar cuando no se puede garantizar una luz homogénea sobre el objeto en estudio, o cuando existe deterioro sobre la imagen fuente causado por suciedad o desgaste o simplemente las condiciones en que se ha capturado la fotografía no resultan óptimas.

7.1. Técnicas globales

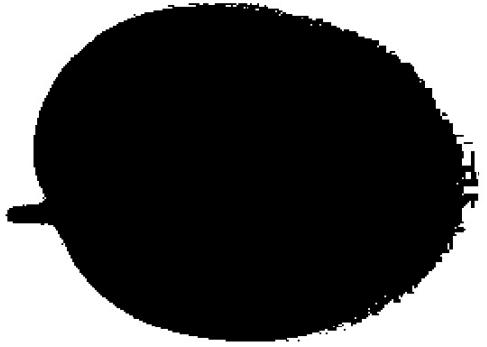
Las técnicas basadas en el histograma para la estimación del umbral son de gran utilidad ya que resultan simples de implementar y rápidas en su procesamiento. Estas requieren que el objeto de interés y el fondo sean claramente definidos en histograma a través de sus modos. En esta sección, se presentan los métodos de agrupamiento de Ridler, el método Otsu [40] y la propuesta de Pun [23, 43] basada en la estimación de la entropía máxima entre regiones del histograma.



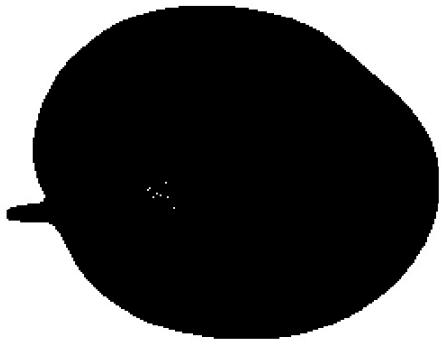
(a) Imagen Original.



(b) Binarizada con Ridler.



(c) Binarizada con Entropia.



(d) Binarizada con OTSU.

Figura 7.4: (a) Imagen original de un mango con fondo uniforme binarizada con los métodos globales (b) Iterativo de Ridler, (c) Entropía y (d) Otsu.

7.1.1. Método global Iterativo

Ridler et al. [44] propusieron un método el cual se basa en la separación por agrupamiento. Si bien su aproximación en el artículo no está basada en el histograma, este se puede explicar haciendo uso de este:

1. Se debe tomar un valor inicial T_{ini} que puede ser el promedio de toda la imagen. Use (7.2):

$$T_{ini} = \frac{\sum_{i=0}^{K-1} h(i) \times i}{\sum_{i=0}^{K-1} h(i)} \quad (7.2)$$

2. Seguidamente, se calcula un una nuevo T_{bajo} haciendo el promedio solo para las intensidades de los píxeles por debajo de umbral inicial T_{ini} y otro valor T_{alto} solo para el promedio de intensidades mayores iguales a este mediante (7.3) y (7.4), respectivamente.

$$T_{bajo} = \frac{\sum_{i=0}^{T_{ini}} h(i) \times i}{\sum_{i=0}^{T_{ini}} h(i)} \quad (7.3)$$

$$T_{alto} = \frac{\sum_{i=T_{ini}+1}^{K-1} h(i) \times i}{\sum_{i=T_{ini}+1}^{K-1} h(i)} \quad (7.4)$$

3. Luego, se determina el nuevo umbral T obteniendo el promedio entre ambos umbrales mediante (7.5):

$$T = (T_{bajo} + T_{alto})/2 \quad (7.5)$$

4. Se evalúa el valor absoluto del cambio ΔT en los umbrales promedio T y el T_{ini} como se indica en (7.6):

$$\Delta T = abs(T - T_{ini}) \quad (7.6)$$

5. El procedimiento se repite hasta lograr que el cambio ΔT sea menor que un valor fijo L o hasta que se realicen un número n de iteraciones definidas.

6. Si $\Delta T \leq L$, se selecciona T como el umbral deseado. De lo contrario, se hace $T_{ini} = T$ y se repite el procedimiento desde el paso 2.

7. Los pixeles que sean mayor iguales al umbral T , conviértelos en 1 y los inferiores llévelos a 0.

7.1.2. Método global de la Entropía

Pun introdujo un método para estimar el umbral. Su concepto básico es la definición de un coeficiente de anisotropía, que está relacionado con la asimetría del histograma de escala de grises. Este método es numéricamente fácil de implementar [43]. El algoritmo se puede realizar de la siguiente forma:

1. Genere el histograma de probabilidad de toda la imagen. Esto se hace para una representación en escala de grises.
2. Itere sobre cada posible valor del umbral T . Si la imagen es de 8 bits, es desde 0 hasta 255 ($[0, K-1]$ donde $K = 2^8$).
3. Para cada umbral estime la probabilidad de la región P_b de background y P_f de foreground usando 7.7 y 7.8, respectivamente. Nótese que también $P_f = 1 - P_b$.

$$P_b = \frac{\sum_{i=0}^T h(i)}{\sum_{i=0}^{K-1} h(i)} \quad (7.7)$$

$$P_f = \frac{\sum_{i=T+1}^{K-1} h(i)}{\sum_{i=0}^{K-1} h(i)} \quad (7.8)$$

4. Calcule la entropía como se muestra en (7.9):

$$H = -[P_b \log_2(P_b) + P_f \log_2(P_f)] \quad (7.9)$$

$$H = H_b + H_f$$

Teniendo en cuenta que:

$$H_b = \begin{cases} -P_b \log_2(P_b) & \text{si } P_b > 0, \\ 0 & \text{de lo contrario.} \end{cases}$$

$$H_f = \begin{cases} -P_f \log_2(P_f) & \text{si } P_f > 0, \\ 0 & \text{de lo contrario.} \end{cases}$$

5. Seleccione el umbral T que generó la mayor entropía H como el umbral global. Si hay varios menores e iguales, tome el primero.

$$T = \arg \max_t [(H_b(t) + H_f(t))]$$

6. Los pixeles que sean mayor iguales al umbral T , conviértalos en 1 y los inferiores llévelos a 0.

7.1.3. Método global de Otsu

En 1979 el investigador japonés Nobuyuki Otsu [40] diseñó una estrategia estadística que no requiere intervención humana la cual utiliza la varianza como medida de la dispersión de los niveles de gris para determinar un único umbral adecuado global. El método de Otsu se usa ampliamente en el reconocimiento de patrones, la binarización de documentos y la visión por computadora.

El algoritmo se explica así:

1. Genere el histograma de probabilidad de toda la imagen. Esto se hace para una representación en escala de grises.
2. Itere sobre cada posible valor del umbral T . Si la imagen es de 8 bits, es desde 0 hasta 255 ($[0, K-1]$ donde $K = 2^8 - 1$).
3. Para cada umbral calcule el peso W_b de background y W_f de foreground usando 7.10 y 7.11, respectivamente.

$$W_b = \frac{\sum_{i=0}^T h(i)}{\sum_{i=0}^{K-1} h(i)} \quad (7.10)$$

$$W_f = \frac{\sum_{i=T+1}^{K-1} h(i)}{\sum_{i=0}^{K-1} h(i)} \quad (7.11)$$

4. Paralelamente al paso 3, estime el promedio de cada región del histograma, usando las expresiones μ_b e μ_f , mostradas en 7.12 y 7.13, respectivamente.

$$\mu_b = \frac{\sum_{i=0}^T h(i) \times i}{\sum_{i=0}^T h(i)} \quad (7.12)$$

$$\mu_f = \frac{\sum_{i=T+1}^{K-1} h(i) \times i}{\sum_{i=T+1}^{K-1} h(i)} \quad (7.13)$$

5. Con cada valor de peso W y promedio μ , tanto del background como de foreground, determine la varianza entre clases para cada umbral. Use 7.14:

$$\text{varianza}_{\text{Between}} = W_b W_f (\mu_b - \mu_f)^2 \quad (7.14)$$

6. Seleccione el umbral T que generó la mayor varianza entre clases como el umbral global.

La varianza entre clases (también conocida como varianza interclase) es una medida estadística que indica la variación de los datos entre diferentes grupos o clases. Se utiliza comúnmente en el análisis de la varianza (ANOVA) para determinar si hay diferencias significativas entre grupos. Una varianza entre clases alta quiere decir que los grupos son ampliamente diferentes (heterogéneos).

Método	Tolerancia al ruido	Preservación de Bordes	Rapidez de Ejecución	Facilidad de implementación
Ridler	Media	Buena	Media	Alta
Entropía	Media-alta	Buena	Baja	Baja
Otsu	Media	Buena	Alta	Media

Cuadro 7.1: Comparación de algunos de los métodos globales.

- Los pixeles que sean mayor iguales al umbral T , conviértalos en 1 y los inferiores llévelos a 0.

Alternativamente, el método de Otsu se puede realizar usando la varianza dentro de la clase. Para esto reemplace los siguientes pasos:

- Paralelamente al paso 3, estime la varianza del background y del foreground del histograma, usando 7.15 y 7.16, respectivamente:

$$\sigma_b^2 = \frac{\sum_{i=0}^T h(i) \times (i - \mu_b)^2}{\sum_{i=0}^T h(i)} \quad (7.15)$$

$$\sigma_f^2 = \frac{\sum_{i=T+1}^{K-1} h(i) \times (i - \mu_f)^2}{\sum_{i=T+1}^{K-1} h(i)} \quad (7.16)$$

- Con cada valor de peso w y la varianza, tanto del background como de foreground, determine la varianza entre dentro de la clase para cada umbral. Use 7.17.

$$varianza_{within} = W_f \sigma_b^2 - W_f \sigma_{ff}^2 \quad (7.17)$$

La varianza dentro de las clases, también conocida como varianza intraclasa, es una medida estadística que se utiliza en análisis de varianza (ANOVA) para evaluar la variación dentro de un grupo o clase en un conjunto de datos. Una varianza dentro de la clase baja indica que los valores dentro de cada grupo o clase son muy similares entre sí y están muy cerca de la media del grupo. Esto sugiere que hay poca variabilidad en los datos dentro de cada grupo y que los valores tienden a estar muy cercanos entre sí.

- Seleccione el umbral T que generó la menor varianza dentro de las clases como el umbral global.

7.1.4. Comparación de los Métodos Globales Presentados

En la Tabla 7.1 se comparan los métodos adaptivos presentados en este capítulo:

7.1.5. Ejemplo en clase: Binarización Método Ridler

Para la imagen de 3 bits mostrada a en la Figura 7.5 determine el umbral mediante el método de Ridler.

2	0	5	7	6
3	1	0	5	4
2	4	2	5	4
1	5	2	7	6
1	5	6	6	6

Figura 7.5: Imagen de 3 bits a binarizar.

Solución

Primero se realiza el histograma de frecuencias como se muestra en la Tabla 7.2:

Intensidad	Frecuencia $h(i)$	Probabilidad $p(i)$
0	2	0.08
1	3	0.12
2	4	0.16
3	1	0.04
4	3	0.12
5	5	0.20
6	5	0.20
7	2	0.08

Cuadro 7.2: Histograma de Frecuencias y Probabilidad de la imagen.

El valor inicial del umbral T_{ini} se calcula utilizando (7.18) el cual corresponde al promedio de todas las intensidades de la imagen:

$$T_{ini} = \frac{\sum_{i=0}^{K-1} h(i) \times i}{\sum_{i=0}^{K-1} h(i)} \quad (7.18)$$

$$T_{ini} = \frac{0 \times 2 + 1 \times 3 + 2 \times 4 + 3 \times 1 + 4 \times 3 + 5 \times 5 + 6 \times 5 + 7 \times 2}{2 + 3 + 4 + 1 + 3 + 5 + 5 + 2} = \frac{95}{25} = 3.8$$

Para la primera iteración, para la sumatoria se usa en inferior ya que $3 \leq 3.8$ y es el entero más cercano en el histograma que cumple:

$$T_{bajo} = \frac{\sum_{i=0}^{T_{ini}} h(i) \times i}{\sum_{i=0}^{T_{ini}} h(i)} \quad (7.19)$$

$$T_{bajo} = \frac{\sum_{i=0}^3 h(i) \times i}{\sum_{i=0}^3 h(i)} = \frac{0 \times 2 + 1 \times 3 + 2 \times 4 + 3 \times 1}{2 + 3 + 4 + 1} = \frac{14}{10} = 1.4$$

$$T_{alto} = \frac{\sum_{i=T_{ini}+1}^{K-1} h(i) \times i}{\sum_{i=T_{ini}+1}^{K-1} h(i)} \quad (7.20)$$

$$T_{alto} = \frac{\sum_{i=4}^7 h(i) \times i}{\sum_{i=4}^7 h(i)} = \frac{4 \times 3 + 5 \times 5 + 6 \times 5 + 7 \times 2}{3 + 5 + 5 + 2} = \frac{81}{15} = 5.4$$

$$T = (T_{bajo} + T_{alto})/2 \quad (7.21)$$

$$T = \frac{1.4 + 5.4}{2} = \frac{6.8}{2} = 3.4$$

$$\Delta T = \text{abs}(T - T_{ini}) \quad (7.22)$$

$$\Delta T = \text{abs}(3.4 - 3.8) = 0.4$$

Como el valor de $\Delta T = 0.4$ no es menor que el umbral $L = 0.001$ ($\Delta T \leq L$), entonces se continúa con la siguiente iteración con $T_{ini} = T$ es decir, $T_{ini} = 3.4$

Para la segunda iteración, para la sumatoria se usa en inferior ya que $3 \leq 3.4$ y es el entero más cercano que cumple la condición:

$$T_{bajo} = \frac{\sum_{i=0}^3 h(i) \times i}{\sum_{i=0}^3 h(i)} = \frac{0 \times 2 + 1 \times 3 + 2 \times 4 + 3 \times 1}{2 + 3 + 4 + 1} = \frac{14}{10} = 1.4$$

$$T_{alto} = \frac{\sum_{i=4}^7 h(i) \times i}{\sum_{i=4}^7 h(i)} = \frac{4 \times 3 + 5 \times 5 + 6 \times 5 + 7 \times 2}{3 + 5 + 5 + 2} = \frac{81}{15} = 5.4$$

$$T = (T_{bajo} + T_{alto})/2 \quad (7.23)$$

$$T = \frac{1.4 + 5.4}{2} = \frac{6.8}{2} = 3.4$$

$$\Delta T = \text{abs}(T - T_{ini}) \quad (7.24)$$

$$\Delta T = \text{abs}(3.4 - 3.4) = 0$$

Como el valor de $\Delta T = 0$ sí es menor que el umbral $L = 0.001$ ($\Delta T \leq L$), entonces se termina el algoritmo y se lleva al T al entero más próximo $T = 3.4 \simeq 3$

Iteración	T_{ini}	T_{bajo}	T_{alto}	T	ΔT
1	3.8	1.4	5.4	3.4	0.4
2	3.4	1.4	5.4	3.4	0

Cuadro 7.3: Resultados iterativos del método de Ridler.

El proceso iterativo se detuvo después de la segunda iteración ya que el cambio en el umbral ΔT fue de 0, lo cual es menor que el criterio de parada ajustado ($\Delta T \leq 0.001$). Esto significa que el algoritmo convergió rápidamente al mismo umbral óptimo de 3.4, redondeando para uso práctico, sería $T = 3$.

Donde se cumple que:

$$B = I \geq 3$$

2	0	5	7	6
3	1	0	5	4
2	4	2	5	4
1	5	2	7	6
1	5	6	6	6

(a) Matriz de la imagen binarizada por Ridler.

0	0	1	1	1
1	0	0	1	1
0	1	0	1	1
0	1	0	1	1
0	1	1	1	1

(b) Matriz binarizada por Ridler.

Figura 7.6: Imagen umbralizada por el método Ridler para valor $T = 3$.

7.1.6. Ejemplo en clase: Binarización Método de la Entropía

Para la imagen de 3 bits mostrada a en la Figura 7.5 determine el umbral mediante el método de Entropía.

Solución

Para aplicar el método de la entropía para determinar el umbral óptimo en una imagen, se deben seguir los pasos del algoritmo que se ha descrito. Sin embargo, para realizar estos cálculos, se necesita una vez más el histograma de la imagen, que es una representación en escala de grises. Una vez se obtiene el histograma, se iterar sobre cada posible valor del umbral desde cero hasta siete (0-7).

Para el cálculo de la entropía en función del umbral T de 0 a 7, se sigue el proceso.

Para cada umbral T de 0 a 7, se calculan las cantidades usando los histogramas divididos mostrados desde la Figura 7.9 hasta la Figura 7.16:

$$P_b = \frac{\sum_{i=0}^T h(i)}{\sum_{i=0}^{K-1} h(i)} \quad P_f = \frac{\sum_{i=T+1}^{K-1} h(i)}{\sum_{i=0}^{K-1} h(i)}$$

Donde:

- P_b : Probabilidad de fondo.
- P_f : Probabilidad de primer plano.
- $H_b = -P_b \log_2(P_b)$ si $P_b > 0$, de lo contrario $H_b = 0$.
- $H_f = -P_f \log_2(P_f)$ si $P_f > 0$, de lo contrario $H_f = 0$.
- $H = H_b + H_f$: Entropía total.

Para $T = 0$:

$$P_{b_0} = \frac{\sum_{i=0}^0 h(i)}{\sum_{i=0}^7 h(i)} = \frac{2}{25} = 0.08 \quad P_{f_0} = 1 - P_{b_0} = 1 - 0.08 = 0.92$$

$$H_{b_0} = -0.08 \log_2(0.08) \approx 0.2915 \quad H_{f_0} = -0.92 \log_2(0.92) \approx 0.1107$$

$$H_0 = 0.2915 + 0.1107 = 0.4022$$

Para $T = 1$:

$$P_{b_1} = \frac{\sum_{i=0}^1 h(i)}{\sum_{i=0}^7 h(i)} = \frac{2+3}{25} = \frac{5}{25} = 0.20 \quad P_{f_1} = 1 - P_{b_1} = 1 - 0.20 = 0.8$$

$$H_{b_1} = -0.20 \log_2(0.20) \approx 0.4644 \quad H_{f_1} = -0.80 \log_2(0.80) \approx 0.2575$$

$$H_1 = 0.4644 + 0.2575 = 0.7219$$

De manera similar se procede para los demás umbrales. En la Tabla 7.4 se presentan los cálculos de la entropía total H para determinar el umbral óptimo T entre 0 y 7:

	0	1	2	3	4	5	6	7
P_b	0.08	0.2	0.36	0.4	0.52	0.72	0.92	1
P_f	0.92	0.8	0.64	0.6	0.48	0.28	0.08	0
H_b	0.2915	0.4644	0.5306	0.5288	0.4906	0.3412	0.1107	0
H_f	0.1107	0.2575	0.4121	0.4422	0.5083	0.5142	0.2915	0
H	0.4022	0.7219	0.9427	0.9710	0.9988	0.8554	0.4022	0

Cuadro 7.4: Resultados iterativos del método de la entropía.

Se observa que la entropía aumenta con valores de umbral bajos hasta alcanzar un máximo, después del cual comienza a disminuir. Esto sugiere que un umbral $T = 4$ es donde la entropía es máxima, considerándose así el lugar óptimo para distinguir entre el fondo y el primer plano en la imagen según este método. Cabe destacar que, aunque un umbral que maximiza la entropía no garantiza siempre la mejor segmentación visual, sirve como un indicador del punto donde la división de los datos de píxeles divididos en dos grupos proporciona la mayor cantidad de información.

Cuando se calcula la entropía y se halla una probabilidad de cero para una de las clases (fondo o primer plano), no se puede evaluar directamente el $\log(0)$ ya que no está definido y tiende a menos infinito. Este escenario plantea la pregunta de cómo manejar correctamente tales casos en el cálculo de la entropía. En teoría de la información, se sortea este problema considerando que el término correspondiente H en el cálculo de la entropía es cero cuando la probabilidad p asociada es también cero. La razón detrás de esto es que si un evento nunca ocurre, la cantidad total de información esperada es nula, lo que se sustenta matemáticamente dado que el límite de $p \log(p)$ cuando p tiende a cero es igual a cero. Por lo anterior, para implementaciones prácticas, especialmente en la programación de los algoritmos, si la probabilidad p es cero, entonces se establece que el término $p \log_2(p)$ también es cero para la entropía.

Donde se cumple que:

$$B = I \geq 4$$

2	0	5	7	6
3	1	0	5	4
2	4	2	5	4
1	5	2	7	6
1	5	6	6	6

(a) Matriz de la imagen binarizada por Entropía.

0	0	1	1	1
0	0	0	1	1
0	1	0	1	1
0	1	0	1	1
0	1	1	1	1

(b) Matriz binarizada por Entropía.

Figura 7.7: Imagen umbralizada por el método Entropía para valor $T = 4$.

7.1.7. Ejemplo en clase: Binarización Método Otsu

Para la imagen de 3 bits mostrada a en la Figura 7.5 determine el umbral mediante el método de OTSU.

Solución

Como primer paso se realiza el histograma de la imagen a umbralizar:

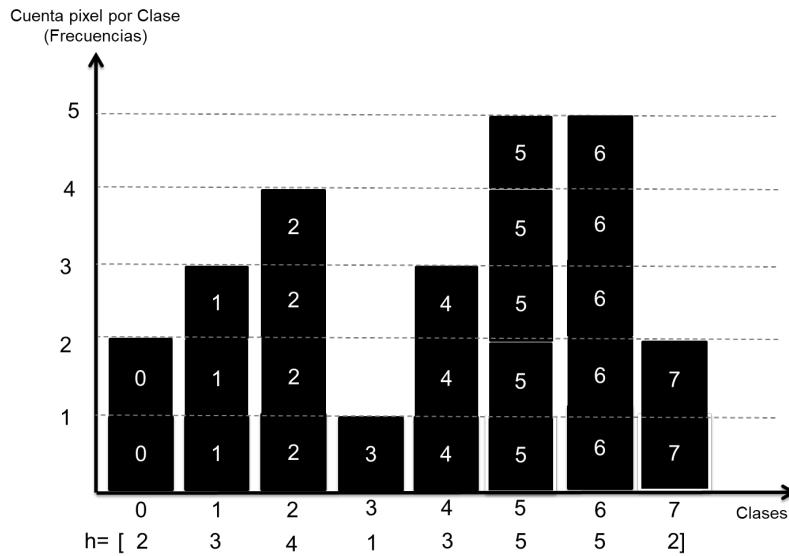


Figura 7.8: Histograma inicial para la imagen de 3 bits.

Se procede a aplicar las ecuaciones de la umbralización para cada valor de T , desde cero hasta siete y sus respectivos μ_b y W_b .

Para $T = 0$:

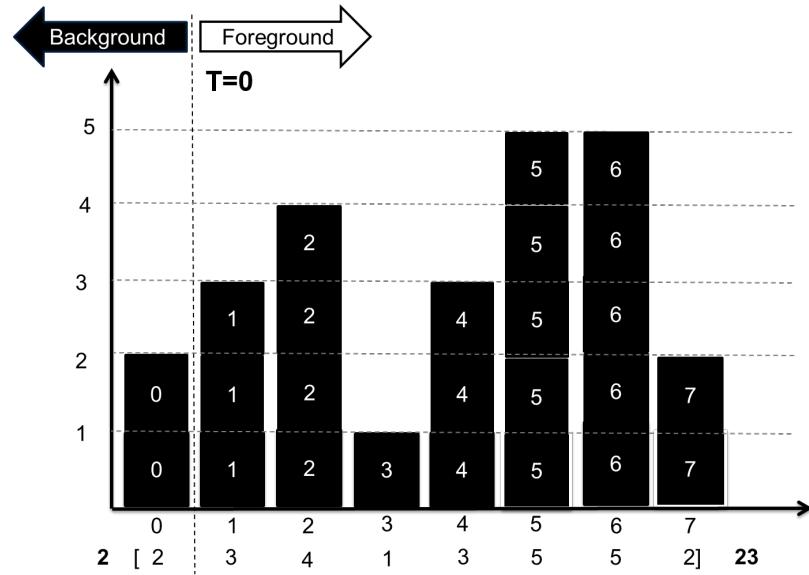


Figura 7.9: Representación del umbral $T = 0$.

$$\begin{aligned}\mu_b &= \frac{0 \times 2}{25} = 0 & \mu_f &= \frac{1 \times 3 + 2 \times 4 + 3 \times 1 + 4 \times 3 + 5 \times 5 + 6 \times 5 + 7 \times 2}{25} = 4.13 \\ W_b &= \frac{2}{25} = 0.08 & W_f &= \frac{3+4+1+3+5+5+2}{25} = 0.92\end{aligned}$$

Para $T = 1$:

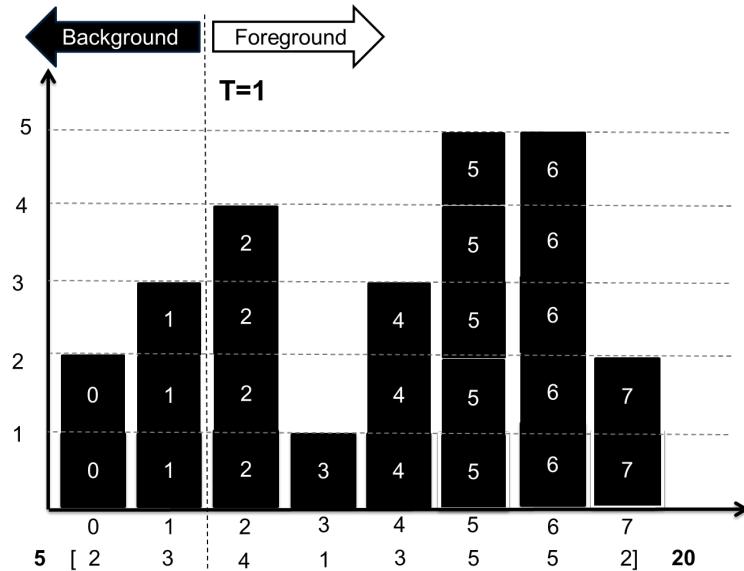


Figura 7.10: Representación del umbral $T = 1$.

$$\begin{aligned}\mu_b &= \frac{0 \times 2 + 1 \times 3}{25} = 0.6 & \mu_f &= \frac{2 \times 4 + 3 \times 1 + 4 \times 3 + 5 \times 5 + 6 \times 5 + 7 \times 2}{25} = 4.6 \\ W_b &= \frac{5}{25} = 0.2 & W_f &= \frac{4+1+3+5+5+2}{25} = 0.8\end{aligned}$$

Para $T = 2$:

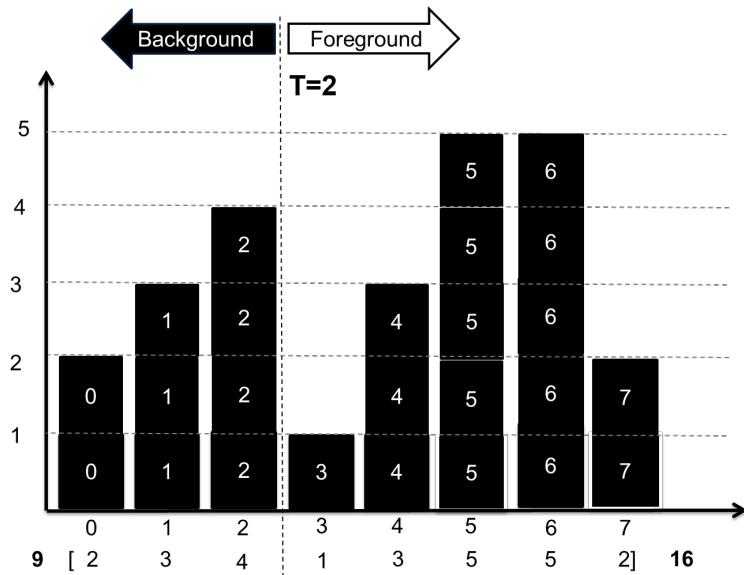


Figura 7.11: Representación del umbral $T = 2$.

$$\begin{aligned} \mu_b &= \frac{0 \times 2 + 1 \times 3 + 2 \times 4}{9} = 1.22 & \mu_f &= \frac{3 \times 1 + 4 \times 3 + 5 \times 5 + 6 \times 5 + 7 \times 2}{16} = 5.25 \\ W_b &= \frac{2+3+4}{25} = 0.36 & W_f &= \frac{1+3+5+5+2}{25} = 0.64 \end{aligned}$$

Para $T = 3$:

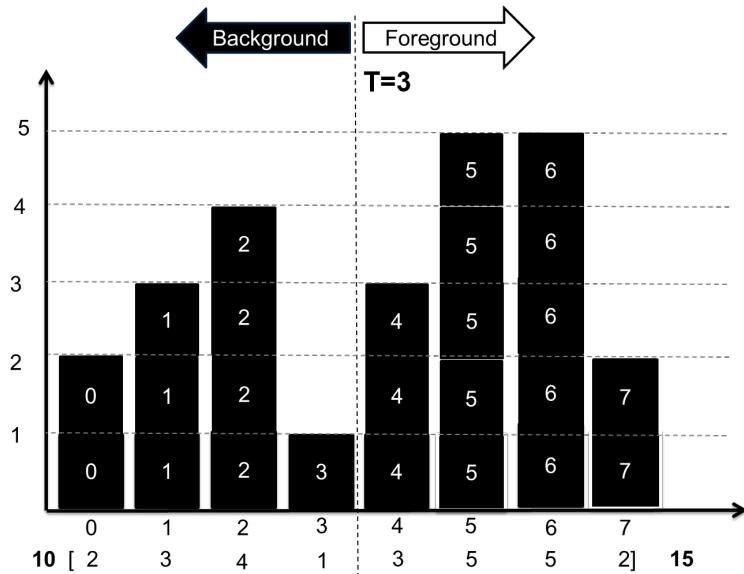


Figura 7.12: Representación del umbral $T = 3$.

$$\begin{aligned} \mu_b &= \frac{0 \times 2 + 1 \times 3 + 2 \times 4 + 3 \times 1}{10} = 1.4 & \mu_f &= \frac{4 \times 3 + 5 \times 5 + 6 \times 5 + 7 \times 2}{15} = 5.4 \\ W_b &= \frac{2+3+4+1}{25} = 0.4 & W_f &= \frac{3+5+5+2}{25} = 0.6 \end{aligned}$$

Para $T = 4$:

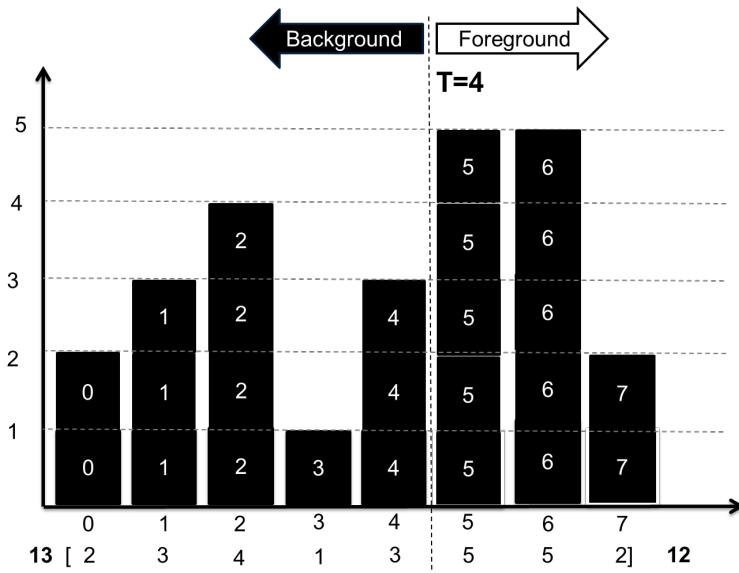


Figura 7.13: Representación del umbral $T = 4$.

$$\mu_b = \frac{0 \times 2 + 1 \times 3 + 2 \times 4 + 3 \times 1 + 4 \times 3}{13} = 2 \quad \mu_f = \frac{5 \times 5 + 6 \times 5 + 7 \times 2}{12} = 5.75$$

$$W_b = \frac{2+3+4+1+3}{25} = 0.52 \quad W_f = \frac{5+5+2}{25} = 0.48$$

Para $T = 5$:

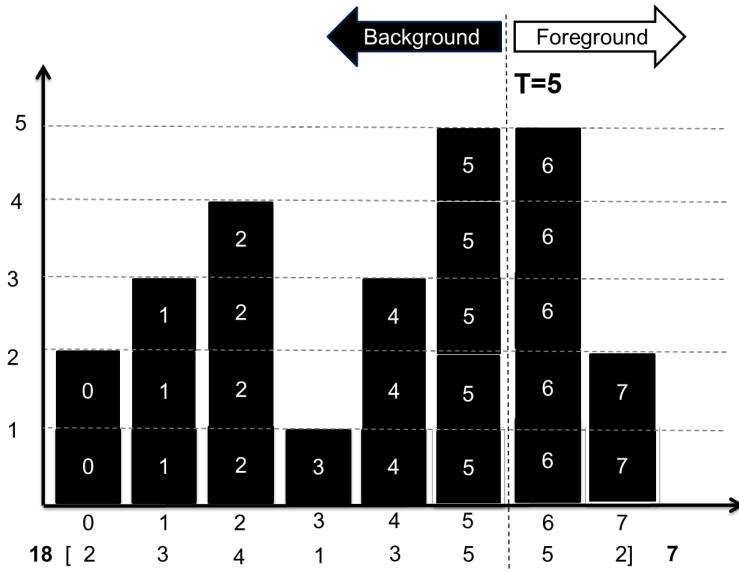


Figura 7.14: Representación del umbral $T = 5$.

$$\mu_b = \frac{0 \times 2 + 1 \times 3 + 2 \times 4 + 3 \times 1 + 4 \times 3 + 5 \times 5}{18} = 2.83 \quad \mu_f = \frac{6 \times 5 + 7 \times 2}{7} = 6.29$$

$$W_b = \frac{2+3+4+1+3+5}{25} = 0.72 \quad W_f = \frac{5+2}{25} = 0.28$$

Para $T = 6$:

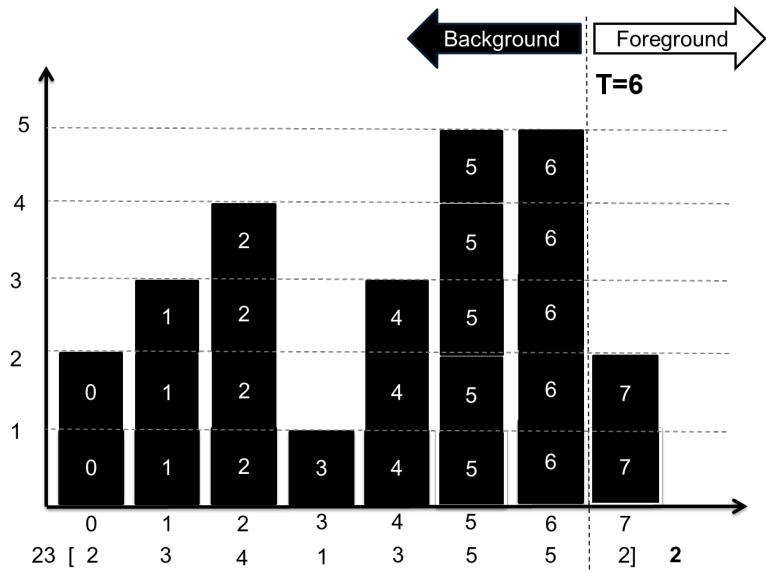


Figura 7.15: Representación del umbral $T = 6$.

$$\begin{aligned}\mu_b &= \frac{0 \times 2 + 1 \times 3 + 2 \times 4 + 3 \times 1 + 4 \times 3 + 5 \times 5 + 6 \times 5}{25} = 3.52 & \mu_f &= \frac{7 \times 2}{25} = 7 \\ W_b &= \frac{2+3+4+23}{25} = 0.92 & W_f &= \frac{2}{25} = 0.08\end{aligned}$$

Para $T = 7$:

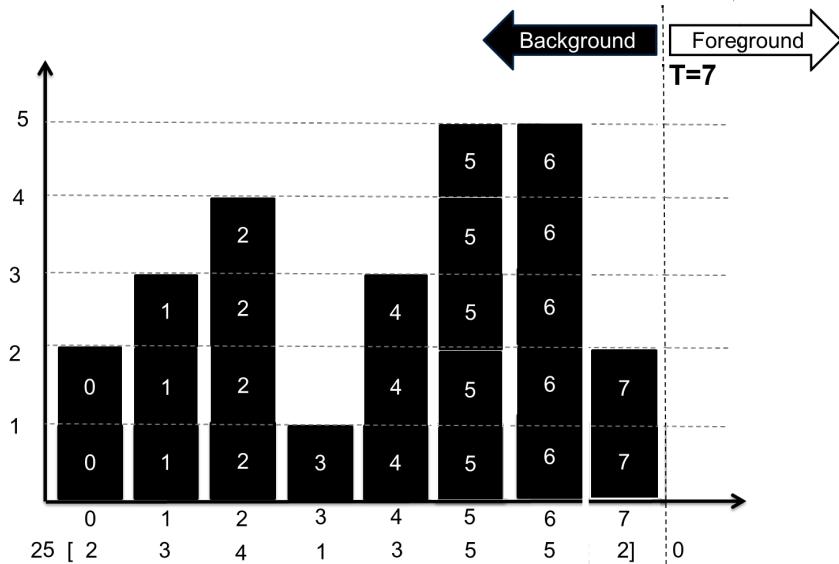


Figura 7.16: Representación del umbral $T = 7$.

$$\begin{aligned}\mu_b &= \frac{0 \times 2 + 1 \times 3 + 2 \times 4 + 3 \times 1 + 4 \times 3 + 5 \times 5 + 6 \times 5 + 7 \times 2}{25} = 3.8 & \mu_f &= 0 \\ W_b &= \frac{2+3+4+25}{25} = 1 & W_f &= 0\end{aligned}$$

Para la solución se reagrupan los resultados obtenidos en el cuadro 7.5 y se selecciona el umbral T correspondiente a la mayor varianza *Between* calculado con la ecuación (7.14):

	T=0	T=1	T=2	T=3	T=4	T=5	T=6	T=7
W_b	0.08	0.2	0.36	0.40	0.52	0.72	0.92	1
μ_b	0.00	0.6	1.22	1.40	2.00	2.83	3.52	3.8
W_f	0.92	0.80	0.64	0.60	0.48	0.28	0.08	0
μ_f	4.13	4.60	5.25	5.40	5.75	6.29	7.00	0
<i>Between</i>	1.26	2.56	3.74	3.84	3.51	2.41	0.89	0

Cuadro 7.5: Resultados obtenidos de pesos y promedios.

Como el mayor número de varianza corresponde a 3.84, se procede a la elección del umbral $T = 3$, leido de la primera fila, para la umbralización global. Esto arroja el resultado mostrado en la Figura 7.17:

Donde se cumple que:

$$B = I \geq 3$$

2	0	5	7	6
3	1	0	5	4
2	4	2	5	4
1	5	2	7	6
1	5	6	6	6

(a) Matriz de la imagen binarizada por OTSU.

0	0	1	1	1
1	0	0	1	1
0	1	0	1	1
0	1	0	1	1
0	1	1	1	1

(b) Matriz binarizada por OTSU.

Figura 7.17: Imagen umbralizada por el método OTSU para valor $T = 3$.

7.1.8. Ejemplo Python

El siguiente ejemplo muestra la aplicación del algoritmo de Otsu para la binarización de imágenes en escala de grises. Este método es útil cuando la imagen presenta un histograma bimodal, es decir, cuando los píxeles pueden clasificarse en dos grupos bien diferenciados: fondo y objeto. El algoritmo determina un umbral óptimo que separa estos dos grupos minimizando la varianza intra-clase o, de manera equivalente, maximizando la varianza inter-clase. En la Fig.7.18 se muestra el proceso de la segmentación del mango usando el umbral de cada capa de color RGB. Finalmente se usa la capa azul.

Solución

```

1
2 #Importación de Librería Básicas
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import sys
6
7 from ip_functions_old import *
8
9 #Abrir Imagen
10 RGB=plt.imread("mango6.jpg")
11 RGB=np.array(RGB);
12 # Factor de Escala S
13 S=2
14 RGB=RGB[1::S,1::S,:]
15 #Seleccionar la capas
16 r=RGB[:, :, 0]
```

```

17 g=RGB[:, :, 1]
18 b=RGB[:, :, 2]
19
20 #Calcular Umbrales con graythresh de OTSU
21
22 #Los umbrales se pueden calcular independientemente con graythresh o otsuthresh
23 Tr=graythresh(r)
24 binr=im2bw(r,Tr)
25 Tg=graythresh(g)
26 bing=im2bw(g,Tg)
27 Tb=graythresh(b)
28 binb=im2bw(b,Tb)
29
30 #Calcular Umbrales con otsuthresh de OTSU
31
32 #Los umbrales se pueden calcular independientemente con graythresh o otsuthresh
33 hr=imhist(r,None,False)
34 hg=imhist(g,None,False)
35 hb=imhist(b,None,False)
36 Tr=otsuthresh(hr)
37 binr=im2bw(r,Tr)
38 Tg=otsuthresh(hg)
39 bing=im2bw(g,Tg)
40 Tb=otsuthresh(hb)
41 binb=im2bw(b,Tb)
42
43 #Graficar Capas y Resultados Umbralización
44 plt.figure(1,figsize=(15,15))
45 plt.subplot(3,3,1)
46 plt.imshow(b,cmap='gray')
47 plt.axis('off')
48 plt.title("Azul")
49 plt.subplot(3,3,2)
50 plt.imshow(g,cmap='gray')
51 plt.axis('off')
52 plt.title("Verde")
53 plt.subplot(3,3,3)
54 plt.imshow(r,cmap='gray')
55 plt.axis('off')
56 plt.title("Rojo")
57
58 plt.subplot(3,3,4)
59 plt.imshow(binr,cmap='gray')
60 plt.axis('off')
61 plt.title(f'Tr={Tr*255:.2f}')
62 plt.subplot(3,3,5)
63 plt.imshow(bing,cmap='gray')
64 plt.axis('off')
65 plt.title(f'Tg={Tg*255:.2f}')
66
67 plt.subplot(3,3,6)
68 plt.imshow(binb,cmap='gray')
69 plt.axis('off')
70 plt.title(f'Tb={Tb*255:.2f}')
71
72
73 ax7=plt.subplot(3,3,7)
74 imhist(r,ax=ax7)
75 ax8=plt.subplot(3,3,8)
76 imhist(g,ax=ax8)
77 ax9=plt.subplot(3,3,9)
78 imhist(b,ax=ax9)
79
80 #Segmentación con Umbral de OTSU
81 Mascara=np.logical_not(binb[:, :, np.newaxis])
82 plt.figure(2,figsize=(15,15))
83 plt.subplot(1,3,1)
84 plt.imshow(RGB)
85 plt.axis('off')
86 plt.title('Original')
87 plt.subplot(1,3,2)

```

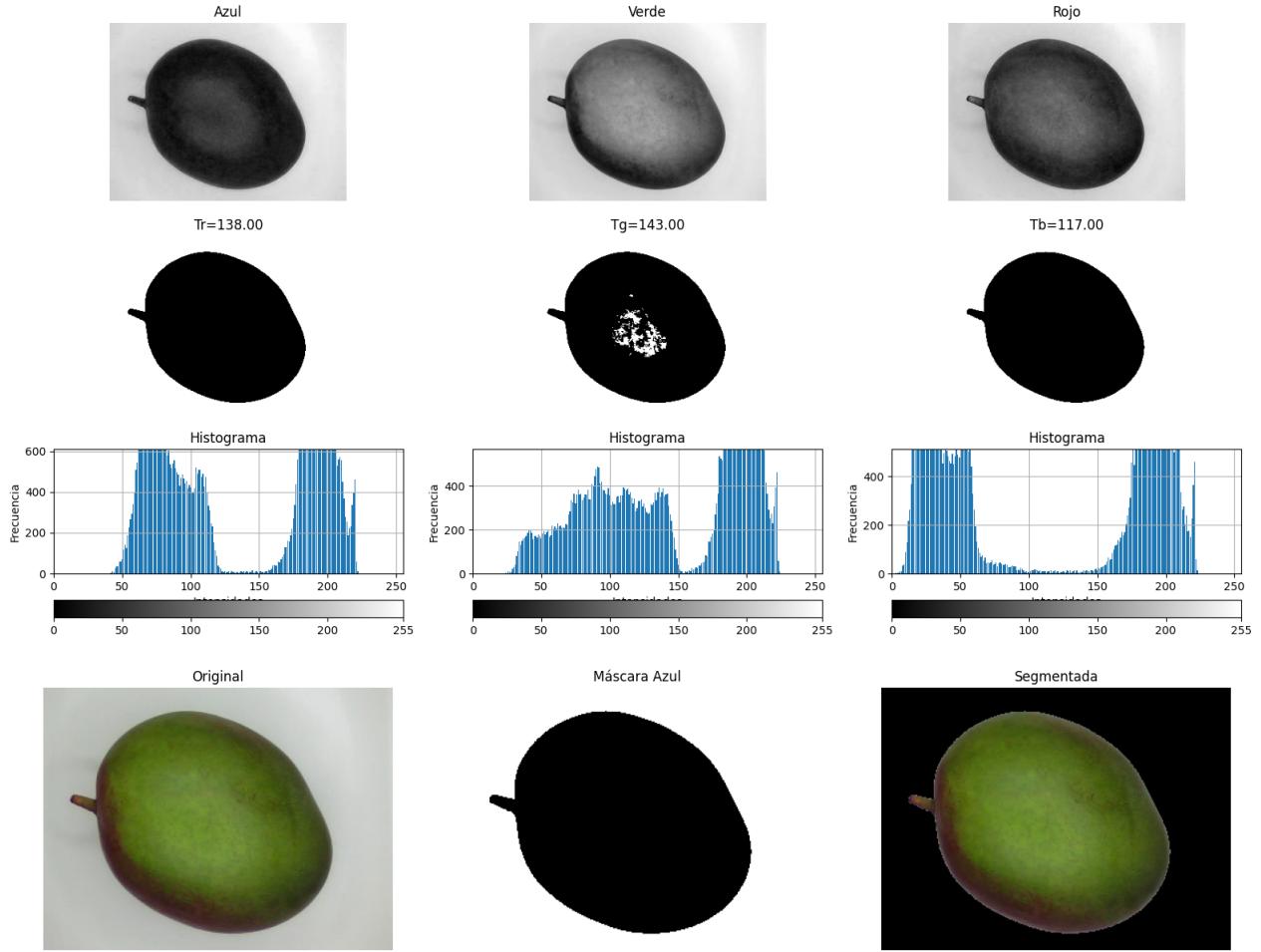


Figura 7.18: Se muestra el proceso detallado de segmentación de un mango a partir de sus componentes de color Rojo (R), Verde (G) y Azul (B). Este se lleva a cabo mediante la aplicación del algoritmo de Otsu, que permite determinar un umbral óptimo para la binarización, separando el objeto de interés (el mango) del fondo. Para este caso se usó el generado por el color azul.

```

88 plt.imshow(binb,cmap='gray')
89 plt.axis('off')
90 plt.title('Máscara Azul')
91 plt.subplot(1,3,3)
92 plt.imshow(Mascara*RGB,cmap='gray')
93 plt.axis('off')
94 plt.title('Segmentada')
95
96 plt.show()

```

7.2. Técnicas Adaptivas

Las imágenes que contienen iluminación no uniforme poseen sombras que no permiten una efectiva binarización haciendo uso de un único umbral como lo propone Otsu. Estas deben binarizarse utilizando algoritmos adaptativos, que realizan el análisis de la vecindad de cada píxel. En esta sección se presentarán los métodos de Niblack, Sauvola y Bradley. En la Figura 7.19 se muestra un documento binarizado usando técnicas globales como Ridler, Otsu y Entropía.

Velocity Estimation From Blurred Image Using DCT

Jimmy Alexander Cortés-Osorio¹, Juan Bernardo

Abstract—There is a growing trend to use a digital camera instead of a regular sensor to measure velocity instead of a regular sensor approach. This paper introduces a new proposal for estimating kinematic quantities, namely, the angle and the relative speed from a single motion blur image using the discrete cosine transform (DCT). Motion blur is a common phenomenon present in images due to the relative movement between the camera and the objects, during sensor exposure to light. Today, this source of kinematic data is mostly dismissed. The introduced technique focuses on cases where the camera moves at a constant linear velocity while the background remains unchanged. 2250 motion blur pictures were shot for the angle experiments and 500 for the speed estimation experiments, in a light and distance controlled environment, using a belt motor slider driven at angles between 0° and 90° and 10 preset speeds. The DCT Hough and DCT Radon results were compared to discrete Fourier transform (DFT) Hough and DFT Radon algorithms for angle estimation. The mean absolute error of the DCT Radon method was 1.25°. In addition, the mean rel

(a) Imagen Original.

Velocity Estimation From Blurred Image Using I

Alexander Cortés-Osorio¹, Juan Bernardo

Abstract—There is a growing trend to use a digital camera instead of a regular sensor to measure velocity instead of a regular sensor approach. This paper introduces a new proposal for estimating kinematic quantities, namely, the angle and the relative speed from a single motion blur image using the discrete cosine transform (DCT). Motion blur is a common phenomenon present in images due to the relative movement between the camera and the objects, during sensor exposure to light. Today, this source of kinematic data is mostly dismissed. The introduced technique focuses on cases where the camera moves at a constant linear velocity while the background remains unchanged. 2250 motion blur pictures were shot for the angle experiments and 500 for the speed estimation experiments, in a light and distance controlled environment, using a belt motor slider driven at angles between 0° and 90° and 10 preset speeds. The DCT Hough and DCT Radon results were compared to discrete Fourier transform (DFT) Hough and DFT Radon algorithms for angle estimation. The mean absolute error of the DCT Radon method was 1.25°. In addition, the mean rel

(c) Binarizada con OTSU.

Velocity Estimation From Blurred Image Using I

Alexander Cortés-Osorio¹, Juan Bernardo

Abstract—There is a growing trend to use a digital camera instead of a regular sensor to measure velocity instead of a regular sensor approach. This paper introduces a new proposal for estimating kinematic quantities, namely, the angle and the relative speed from a single motion blur image using the discrete cosine transform (DCT). Motion blur is a common phenomenon present in images due to the relative movement between the camera and the objects, during sensor exposure to light. Today, this source of kinematic data is mostly dismissed. The introduced technique focuses on cases where the camera moves at a constant linear velocity while the background remains unchanged. 2250 motion blur pictures were shot for the angle experiments and 500 for the speed estimation experiments, in a light and distance controlled environment, using a belt motor slider driven at angles between 0° and 90° and 10 preset speeds. The DCT Hough and DCT Radon results were compared to discrete Fourier transform (DFT) Hough and DFT Radon algorithms for angle estimation. The mean absolute error of the DCT Radon method was 1.25°. In addition, the mean rel

(b) Binarizada con Ridler.

Velocity Estimation From Blurred Image Using I

Alexander Cortés-Osorio¹, Juan Bernardo

Abstract—There is a growing trend to use a digital camera instead of a regular sensor to measure velocity instead of a regular sensor approach. This paper introduces a new proposal for estimating kinematic quantities, namely, the angle and the relative speed from a single motion blur image using the discrete cosine transform (DCT). Motion blur is a common phenomenon present in images due to the relative movement between the camera and the objects, during sensor exposure to light. Today, this source of kinematic data is mostly dismissed. The introduced technique focuses on cases where the camera moves at a constant linear velocity while the background remains unchanged. 2250 motion blur pictures were shot for the angle experiments and 500 for the speed estimation experiments, in a light and distance controlled environment, using a belt motor slider driven at angles between 0° and 90° and 10 preset speeds. The DCT Hough and DCT Radon results were compared to discrete Fourier transform (DFT) Hough and DFT Radon algorithms for angle estimation. The mean absolute error of the DCT Radon method was 1.25°. In addition, the mean rel

(d) Binarizada con Entropía.

Figura 7.19: (a) Imagen original de un documento con sombras, b) binarizada con Ridler, (c) binarizada con Otsu y (d) binarizada con Entropía.

7.2.1. Método Adaptivo de Niblack

El enfoque fue desarrollado por Wayne Niblack [37] quien propuso la determinación del umbral local $T(x, y)$ para cada pixel realizando el promedio de las intensidades vecinas de la ventana, para luego disminuir el valor de la desviación estándar de esta escalada por el parámetro constante K .

Dados una imagen en escala de grises I , un tamaño de ventana $w \times w$, y un parámetro de ajuste K , el algoritmo calcula un umbral adaptativo $T(x, y)$ para cada píxel (x, y) en I como sigue:

1. Para cada píxel (x, y) en la imagen I , definir una ventana local W centrada en (x, y) con tamaño $w \times w$.
2. Calcular la media $\mu(x, y)$ y la desviación estándar de la muestra $\sigma(x, y)$ de los píxeles dentro de W .
3. Calcular el umbral adaptativo $T(x, y)$ para el píxel (x, y) usando la expresión 7.25:

$$T(x, y) = \mu(x, y) + K \cdot \sigma(x, y) \quad (7.25)$$

Los valores típicos para K varían dependiendo de las características específicas de la imagen y el objetivo de la binarización. Sin embargo, se sugieren varios rangos comunes para K :

- **Para imágenes generales:** Un rango común para K es entre -0.2 y -0.5. Estos valores suelen funcionar bien para una amplia gama de imágenes, proporcionando un buen equilibrio entre el contraste y la preservación de detalles.
- **Para textos o documentos:** Para la binarización de textos o documentos escaneados, donde el objetivo es separar claramente el texto (negro) del fondo (blanco), se pueden preferir valores de K más negativos, como -0.5 o incluso hasta -0.7. Esto ayuda a reducir el ruido y mejorar la legibilidad del texto.
- **Para imágenes con bajo contraste:** En casos de imágenes con bajo contraste, donde se desea preservar más detalles, se pueden utilizar valores de K más cercanos a cero o ligeramente positivos.

En general, se debe experimentar con diferentes valores de K y evaluar los resultados es una práctica común para encontrar el ajuste más adecuado para una tarea dada.

1. Binarizar la imagen I aplicando el umbral $T(x, y)$ a cada píxel:

$$B(x, y) = \begin{cases} 1 & \text{si } I(x, y) \geq T(x, y) \\ 0 & \text{si } I(x, y) < T(x, y) \end{cases}$$

Este método se adapta a las variaciones locales de iluminación y textura, permitiendo una binarización efectiva incluso en condiciones de iluminación no uniforme.

7.2.2. Método Adaptivo de Sauvola

De forma similar a lo definido por Nickblack, el método de Sauvola [48] propone un umbral $T(x, y)$ para cada pixel el cual depende de la vecindad empleada además de otros aspectos.

Dados una imagen en escala de grises I , un tamaño de ventana $w \times w$, un parámetro de ajuste K y un valor de normalización R , el algoritmo calcula un umbral adaptativo $T(x, y)$ para cada píxel (x, y) en I como sigue:

1. Para cada píxel (x, y) en la imagen I , definir una ventana local W centrada en (x, y) con tamaño $w \times w$.
2. Calcular la media aritmética (promedio) $\mu(x, y)$ y la desviación estándar de la muestra $\sigma(x, y)$ de los píxeles dentro de W .

3. Calcular el umbral adaptativo $T(x, y)$ para el píxel (x, y) usando la expresión 7.26:

$$T(x, y) = \mu(x, y) \left[1 + K \left(\frac{\sigma(x, y)}{R} - 1 \right) \right] \quad (7.26)$$

donde K es un factor que ajusta la sensibilidad del umbral a la variabilidad local de la intensidad en un valor comprendido entre [0.2,0.5], y R es el valor máximo de la desviación estándar esperado, típicamente 128 para imágenes de 8 bits en escala de grises; es decir, la mitad de su escala máxima.

4. Binarizar la imagen I aplicando el umbral $T(x, y)$ a cada píxel:

$$B(x, y) = \begin{cases} 1 & \text{si } I(x, y) \geq T(x, y) \\ 0 & \text{si } I(x, y) < T(x, y) \end{cases}$$

Este método adapta el umbral a las variaciones locales de iluminación y textura, mejorando la binarización en condiciones de iluminación no uniforme y preservando detalles finos de la imagen. El método de Sauvola tiene la capacidad de adaptarse a las variaciones locales de iluminación y contraste.

7.2.3. Método Adaptivo de Bradley

La idea principal del algoritmo de Bradley [4] es que cada píxel de la imagen se lleve negro (cero) si su brillo es p por ciento más bajo que el brillo promedio de los píxeles circundantes en la ventana, y en la condición opuesta la hace blanco (uno) usando un umbral $T(x, y)$. De manera similar a los métodos anteriores, la operación se hace pixel a pixel recorriendo por ventanas. En el trabajo original, se empleó la técnica de la imagen integral para la umbralización adaptativa de las imágenes. Esta permite calcular rápidamente la suma de intensidades en áreas extensas de la imagen con un costo computacional constante, lo cual es esencial para aplicaciones que requieren procesamiento en tiempo real, como la realidad aumentada. No obstante, en este documento, se ha optado por utilizar una aproximación más simplificada con base en el cálculo de los promedios sobre ventanas deslizantes. Esta decisión se justifica por la simplicidad y menor complejidad de implementación que ofrece el método de ventanas promedio, especialmente en contextos donde la velocidad de procesamiento no es crítica, pero en términos de rendimiento general y capacidad para manejar variaciones dinámicas de iluminación, la técnica original de la imagen integral sigue siendo superior.

Dados una imagen en escala de grises I , un tamaño de ventana $w \times w$, y un porcentaje p , el algoritmo calcula un umbral adaptativo $T(x, y)$ para cada píxel (x, y) en I como sigue:

1. Para cada píxel (x, y) en la imagen I , definir una ventana local W centrada en (x, y) con tamaño $w \times w$.
2. Calcular la media aritmética (promedio) $\mu(x, y)$ de los píxeles dentro de W .
3. Calcular el umbral adaptativo $T(x, y)$ para el píxel (x, y) usando la fórmula:

$$T(x, y) = \mu(x, y) \left[1 - \frac{p}{100} \right] \quad (7.27)$$

donde p es el porcentaje por el cual se reduce la media para establecer el umbral.

4. Binarizar la imagen I aplicando el umbral $T(x, y)$ a cada píxel:

$$B(x, y) = \begin{cases} 1 & \text{si } I(x, y) \geq T(x, y) \\ 0 & \text{si } I(x, y) < T(x, y) \end{cases}$$

Este método se adapta a las variaciones locales de iluminación y textura, permitiendo una binarización efectiva incluso en condiciones de iluminación no uniforme. La selección del tamaño de ventana $w \times w$ y el valor de p pueden ajustarse para optimizar el rendimiento del algoritmo en diferentes tipos de imágenes.

En la Figura 7.20 se presentan algunas muestras de un documento binarizado usando técnicas adaptativas como Nickblack, Sauvola y Bradley.

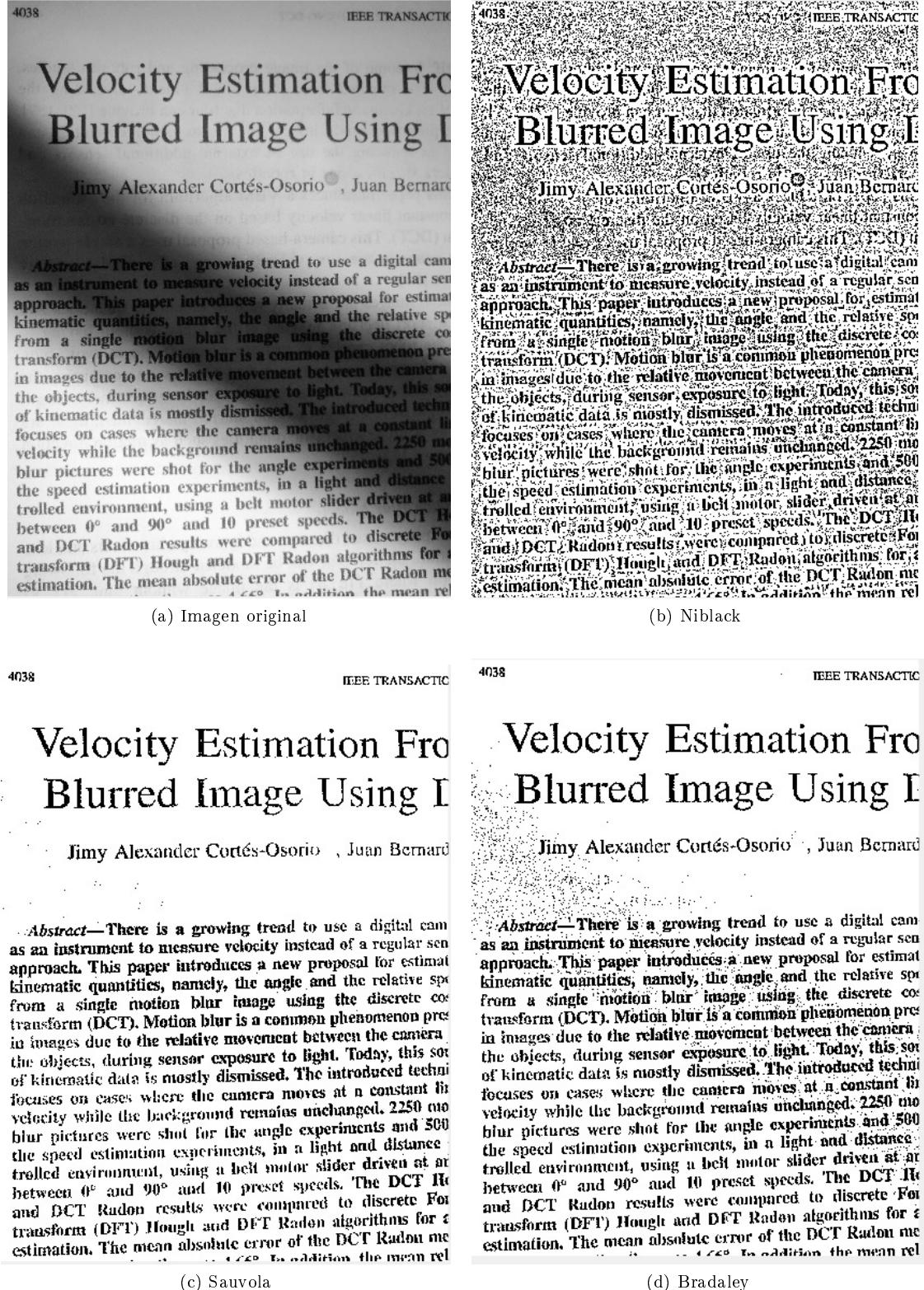


Figura 7.20: (a) Imagen original de un documento con sombras y (b) la versión binarizada con Bradley.

Método	Tolerancia al ruido	Preservación de Bordes	Rapidez de Ejecución	Facilidad de implementación
Niblack	Media-baja	Muy buena	Media	Media
Sauvola	Alta	Suavizado	Baja	Baja
Bradley	Baja	Buena	Alta	Alta

Cuadro 7.6: Comparación de algunos de los métodos adaptivos.

$$I = \begin{bmatrix} 1 & 3 & 1 \\ 5 & 7 & 6 \\ 2 & 4 & 2 \end{bmatrix}$$

Cuadro 7.7: Ejercicio de binarización adaptiva para una imagen de 3 bits.

7.2.4. Comparación de los Métodos Adaptivos Presentados

En la Tabla 7.6 se comparan los métodos adaptivos presentados en este capítulo:

7.2.5. Ejemplo en clase: Binarización Método de Niblack

Realice la binarización adaptativa de la imagen de 3 bits, representada por la matriz 3×3 con un factor de $k = -0.2$. Véase la Tabla 7.7. Utilizando una ventana de 3×3 para el cálculo del umbral, debe extender los bordes de la matriz para asegurar que cada píxel, incluidos los de los bordes, pueda ser evaluados dentro de una ventana completa. El ejercicio solicita mostrar tanto la matriz original como la resultante tras aplicar la binarización adaptativa.

Solución

Inicialmente, es necesario expandir la matriz de la imagen I para permitir que la ventana (Flotante) de correlación se desplace a través de toda la imagen sin alterar su tamaño original. Luego, se calcula la matriz de promedio (de ventana) y, opcionalmente de manera simultánea, la matriz de desviaciones estándar de la muestra para cada ventana local. A continuación, se aplica la fórmula de Niblack, mostrada en (7.28), para obtener la matriz de umbrales T . Finalmente, se utiliza la condición $I \geq T$ para binarizar la imagen, resultando en una matriz donde los píxeles que cumplen o superan el umbral se establecen en 1, y los demás en 0. Véase la Figura 7.21.

$$T(x, y) = \mu(x, y) + K \cdot \sigma(x, y) \quad (7.28)$$

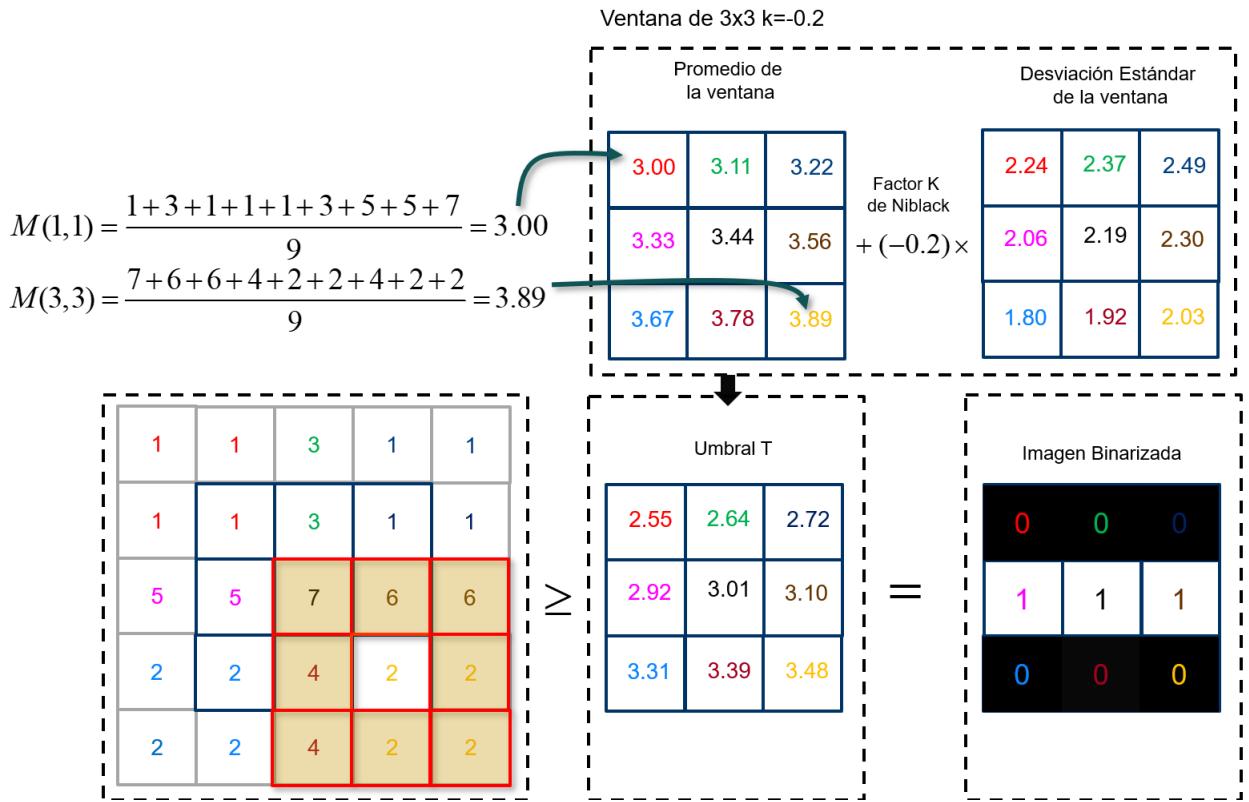


Figura 7.21: Binarización de una imagen de 3×3 y 3 bits mediante el método de Niblack. Se amplía la matriz I , se calculan los promedios y desviaciones estándar locales, y se aplica la fórmula de Niblack para obtener la matriz de umbrales T . Finalmente, se presenta la imagen adaptivamente binarizada.

7.2.6. Ejemplo en clase: Binarización Método de Sauvola

Realice la binarización adaptativa mediante el método de Sauvola de la imagen de 3 bits, representada por la matriz 3×3 con un factor de $k = 0.5$. Véase la Tabla 7.7. Utilizando una ventana de 3×3 para el cálculo del umbral, debe extender los bordes de la matriz para asegurar que cada píxel, incluidos los de los bordes, pueda ser evaluados dentro de una ventana completa. El ejercicio solicita mostrar tanto la matriz original como la resultante tras aplicar la binarización adaptativa.

Solución

Para la binarización adaptativa mediante Sauvola, es necesario preparar la matriz de la imagen. Inicialmente, se amplía la matriz de la imagen I para que la ventana de correlación pueda recorrer la imagen completa sin alterar su tamaño original. Posteriormente, se calculan las matrices de promedio y, si se desea, de desviaciones estándar de cada ventana local de manera simultánea, similar al método de Niblack ya descrito. Finalmente, se usa (7.29), para calcular la matriz de umbrales T , que depende de la media μ y la desviación estándar σ locales. Finalmente, se utiliza la condición $I \geq T$ para binarizar la imagen, resultando en una matriz donde los píxeles que cumplen o superan el umbral se establecen en 1, y los demás en 0. Véase la Figura 7.22.

$$T(x, y) = \mu(x, y) \left[1 + K \left(\frac{\sigma(x, y)}{R} - 1 \right) \right] \quad (7.29)$$

donde $K = 0.5$ y $R = 3$ son parámetros adaptados a imágenes de 3 bits. Estos son escogidos considerando que el rango de desviación estándar será relativamente bajo debido al limitado espectro dinámico de los píxeles, que varían de 0 a 7.

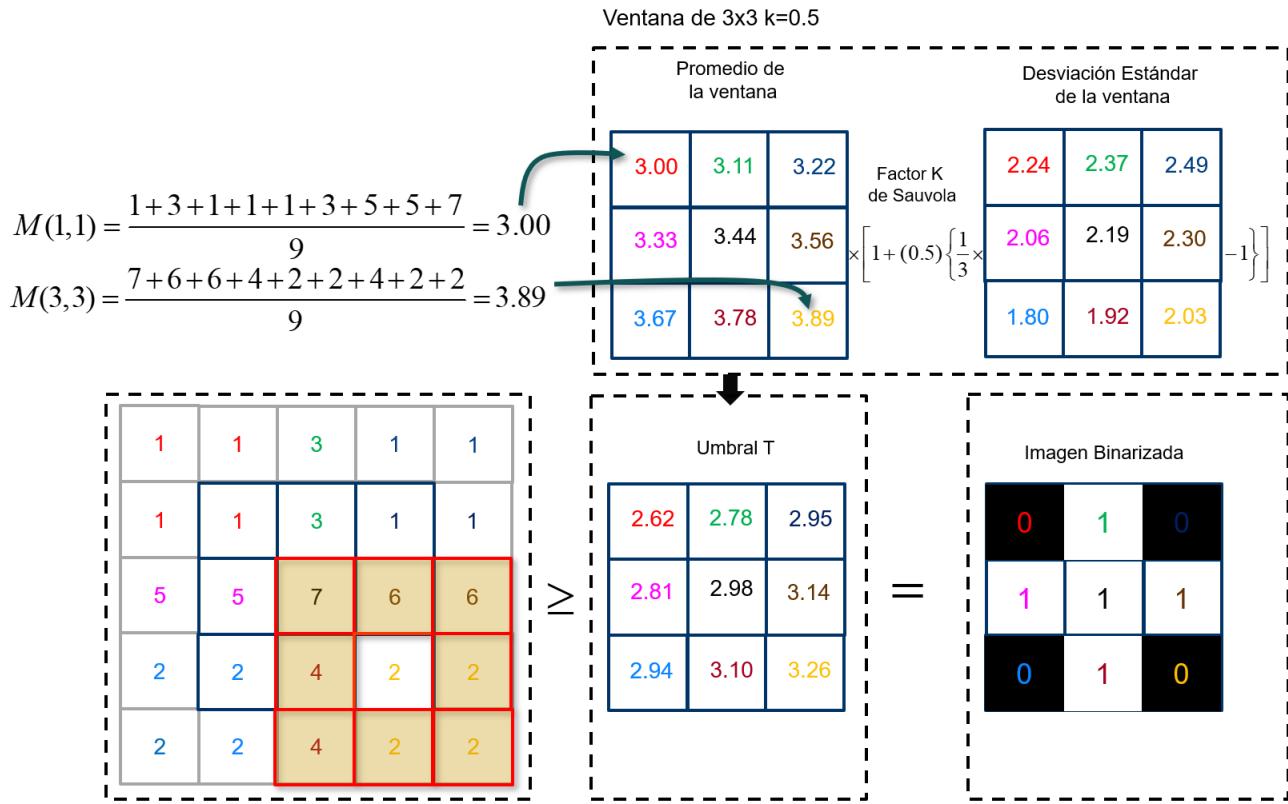


Figura 7.22: Binarización de una imagen de 3×3 y 3 bits mediante el método de Sauvola. Se amplía la matriz I , se calculan los promedios y desviaciones estándar locales, y se aplica la fórmula de Sauvola para obtener la matriz de umbrales T . Finalmente, se presenta la imagen adaptativamente binarizada.

7.2.7. Ejemplo en clase: Binarización Método de Bradley

Realice la binarización adaptativa mediante el método de Bradley para la imagen de 3 bits, representada por la matriz 3×3 con un factor de $P = 15$. Véase la Tabla 7.7. Utilizando una ventana de 3×3 para el cálculo del umbral, debe extender los bordes de la matriz para asegurar que cada píxel, incluidos los de los bordes, pueda ser evaluados dentro de una ventana completa. El ejercicio solicita mostrar tanto la matriz original como la resultante tras aplicar la binarización adaptativa.

Solución

La binarización adaptativa mediante el método de Bradley requiere una preparación inicial de la matriz de la imagen. Se amplía la matriz I para permitir que la ventana de correlación recorra la imagen completa sin alterar su tamaño original. Luego, se calcula la matriz de promedio para multiplicarla por un factor. La matriz de umbrales T se determina usando (7.30):

1.

$$T(x,y) = \mu(x,y) \left[1 - \frac{p}{100} \right] \quad (7.30)$$

donde μ es el promedio local de la intensidad de los píxeles dentro de la ventana, y p es el porcentaje de ajuste del umbral.

Finalmente, se aplica la condición $I \geq T$ para binarizar la imagen. En esta condición, los píxeles cuya intensidad es mayor o igual al umbral se establecen en 1, mientras que los demás se establecen en 0.

Este proceso resulta en una matriz binaria que representa la imagen binarizada adaptativamente. La Figura 7.23

ilustra este proceso.

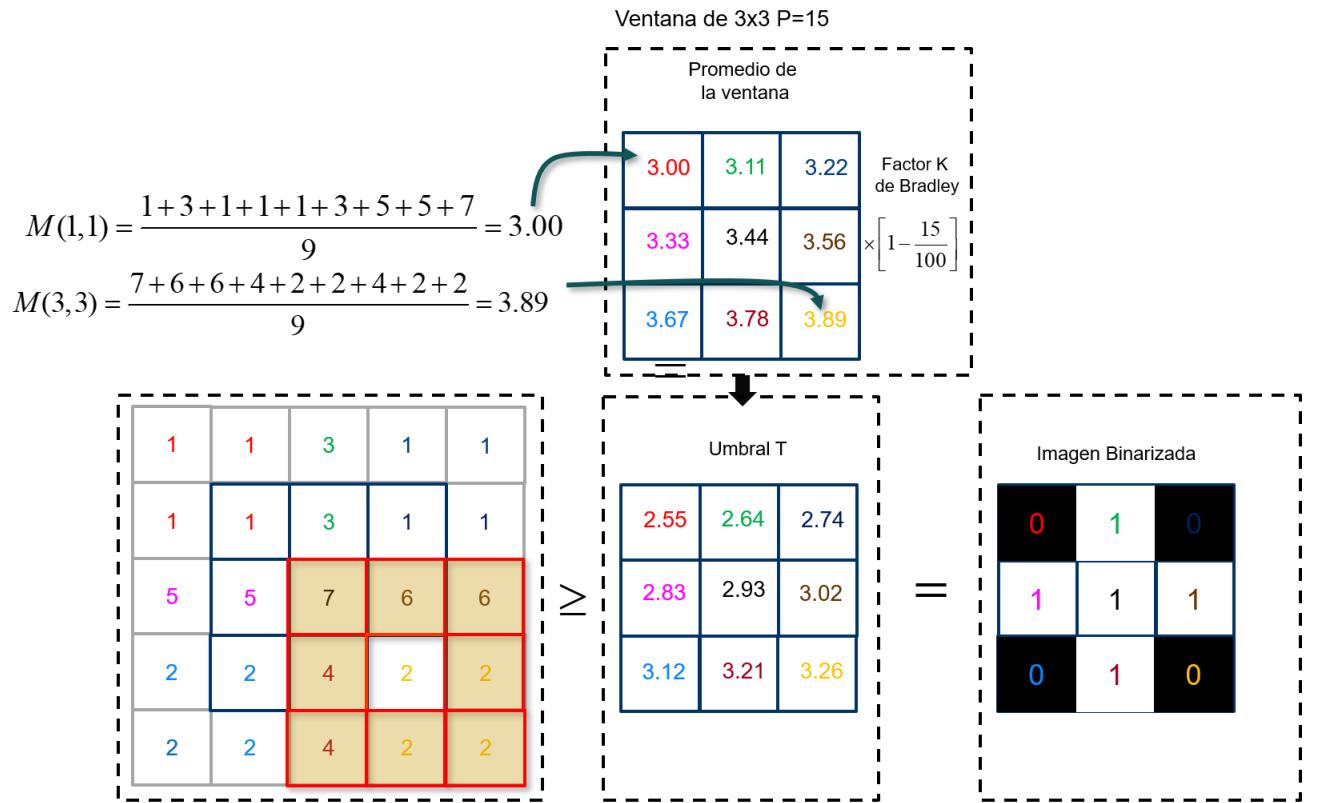


Figura 7.23: Binarización de una imagen de 3×3 y 3 bits mediante el método de Bradley. Se amplía la matriz I , se calculan los promedios y desviaciones estándar locales, y se aplica la fórmula de Bradley para obtener la matriz de umbrales T . Finalmente, se presenta la imagen adaptativamente binarizada.

7.2.8. Ejemplo en Python: comparación de la binarización global y adaptativa

Este código compara la segmentación de una imagen en escala de grises utilizando un método global y uno adaptativo. Primero, se carga y reduce el tamaño de la imagen, luego se convierte a escala de grises. Para la segmentación, se emplea el umbral global de Otsu y el método adaptativo de Bradley, que ajusta el umbral según la intensidad local en una ventana determinada. Finalmente, se muestran las imágenes resultantes junto con la imagen original para visualizar las diferencias entre ambos enfoques.

Solución

```

1 """
2 Este script muestra la comparación de la segmentación de una
3 imagen en escala de grises.
4 """
5 #Importación de Librería Básicas
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import sys
9
10 from ip_functions import *
11
12 # Open image
13 RGB=plt.imread("paper.jpg")
14 RGB=np.array(RGB)
15
16 # Factor de Escala S
17 S=5

```

```

18 RGB=RGB[0::S,0::S,:]
19 #Convertir a escala de grises
20 gris=rgb2gray(RGB)
21
22 #Selección Umbral por OTSU
23 #T=graythresh(gris)
24 #binOTSU=im2bw(gris,T)
25
26
27 #Altermantivamente se puede hacer con imbinarize
28 binOTSU = imbinarize(gris,'global')
29
30 #Selección Umbral por BRADLEY
31
32 V=[13,13] #Ventana a usar
33 P=0.15    # Porcentaje del promedio
34
35 #Tm=adaptthresh(gris,P,V) #Matriz de umbrales
36 #binBRADLEY=im2bw(gris,Tm)
37
38 #Altermantivamente se puede hacer con imbinarize
39 binBRADLEY = imbinarize(gris,'adaptive','Sensitivity',P,'WindowSize',V)
40
41
42 #Comparación OTSU vs BRADLEY
43 plt.figure(figsize = (15,15))
44 plt.subplot(1,3,1)
45 plt.imshow(gris, cmap='gray')
46 plt.title("Imagen Referencia")
47 plt.axis('off')
48
49 plt.subplot(1,3,2)
50 plt.imshow(binOTSU, cmap='gray')
51 plt.title("Global OTSU")
52 plt.axis('off')
53
54 plt.subplot(1,3,3)
55 plt.imshow(binBRADLEY, cmap='gray')
56 plt.title("-Adaptivo BRADLEY")
57 plt.axis('off')
58 plt.show()

```

La Fig.7.24 muestra la comparación entre la segmentación global mediante el método de Otsu y la segmentación adaptativa con el método de Bradley, aplicados a una imagen en escala de grises. En la primera imagen se presenta la referencia original, mientras que en las siguientes se observan los resultados de cada técnica, destacando las diferencias en la detección de regiones umbralizadas. La implementación de estas funciones se detalla en el capítulo 13.

7.3. Funciones

Las siguientes funciones tienen relación con el tema abordado en este capítulo:

1. **otsuthresh(h):** calcula el umbral óptimo a partir de un histograma **h** de niveles de gris (256 valores), utilizando el método de Otsu que maximiza la varianza entre clases y retorna el índice del umbral óptimo (entre 0 y 255).
2. **graythresh(I):** estima automáticamente el umbral óptimo para binarizar una imagen en escala de grises **I** utilizando el método de Otsu aplicado al histograma normalizado de la imagen. La imagen **I** debe estar en el rango [0, 255].
3. **im2bw(I, T):** convierte la imagen en escala de grises **I** a una imagen binaria aplicando un umbral **T** (entre 0 y 1). Todos los valores mayores o iguales a **T** se asignan a 1 (blanco) y los menores a 0 (negro). La imagen de entrada debe estar normalizada entre 0 y 1.
4. **adaptthresh(I, P=0.15, V=(15, 15)):** determina una matriz de umbrales locales para la imagen **I** mediante

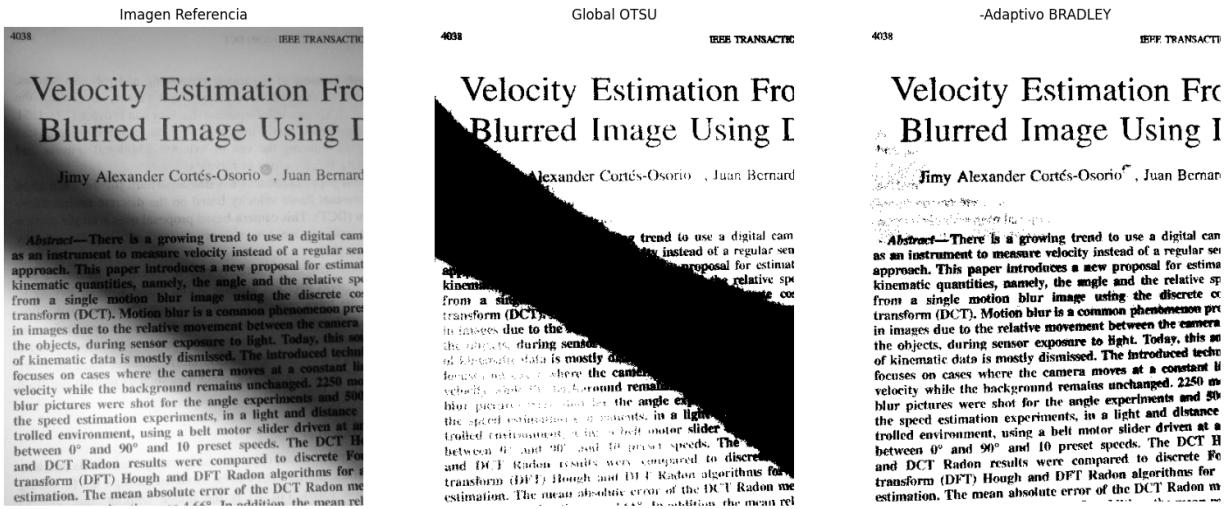


Figura 7.24: La Figura muestra la segmentación de una imagen en escala de grises utilizando el umbral global de Otsu y el método adaptativo de Bradley. Se observa cómo Otsu aplica un único umbral, mientras que Bradley ajusta localmente el umbral según la iluminación.

binarización adaptativa, usando una sensibilidad $P \in [0, 1]$ (por defecto 0.15) y un tamaño de ventana V como tupla (ancho, alto) para definir el vecindario.

5. `imbinarize(I, modo='global', T=None, P=0.15, V=(15, 15))`: convierte una imagen en escala de grises I a binaria utilizando umbralización global (`modo='global'`), adaptativa (`modo='adaptive'`) o un umbral escalar T definido por el usuario. En modo adaptativo, se puede ajustar la sensibilidad P y el tamaño de vecindario V .

7.4. Preguntas

7.4.1. Preguntas sobre hechos indicados en el texto

1. ¿En qué consiste el proceso de binarización de imágenes? Cuál es la diferencia con umbralización?
2. ¿Cuáles son los principales enfoques para estimar el umbral en la binarización?
3. ¿Cuál es la ventaja de los métodos adaptativos sobre los globales sus las posibles desventajas?
4. ¿Para qué tipo de aplicaciones es útil el proceso de binarización?
5. ¿Qué hace más útil las técnicas de binarización con base en el histograma?

7.4.2. Preguntas de análisis y comprensión con base en el texto

1. ¿Por qué la binarización permite reducir información no relevante en una imagen? ¿En qué tipos de aplicaciones esto es útil?
2. ¿Cuál es la desventaja de utilizar un umbral global en imágenes con variaciones de iluminación? ¿Cómo lo solucionan los métodos adaptativos?
3. ¿Cómo seleccionaría el tamaño de ventana apropiado en los métodos de binarización adaptativa? ¿Qué efectos tiene usar una ventana muy pequeña o muy grande?

4. Consulte el método de las sumas integrales para agilizar la determinación del promedio de subventanas dentro de una matriz.

7.4.3. Ejercicios numéricos

1. Para la matriz de la imagen de 5×5 pixels de 4 bits mostrada en 7.31, realice de forma manuscrita la binarización mediante el método de OTSU. Explique cada paso.

$$\begin{bmatrix} 0 & 2 & 4 & 12 & 3 \\ 11 & 12 & 13 & 14 & 11 \\ 10 & 6 & 14 & 5 & 9 \\ 3 & 12 & 13 & 15 & 10 \\ 3 & 2 & 4 & 11 & 1 \end{bmatrix} \quad (7.31)$$

2. Para la matriz de la imagen de 5×5 pixels de 4 bits mostrada en 7.31, realice de forma manuscrita la binarización global mediante el método de Ridler. Explique cada paso.
3. Para la matriz de la imagen de 5×5 pixels de 4 bits mostrada en 7.31, realice de forma manuscrita la binarización global mediante el método de la Entropía. Explique cada paso.
4. Para la matriz de la imagen de 5×5 pixels de 4 bits mostrada en 7.31, realice de forma manuscrita la binarización global mediante el método de la Niblack. Use una ventana de 3×3 y replique los bordes. Explique cada paso.
5. Para la matriz de la imagen de 5×5 pixels de 4 bits mostrada en 7.31, realice de forma manuscrita la binarización global mediante el método de la Sauvola. Use una ventana de 3×3 y replique los bordes. Explique cada paso.
6. Para la matriz de la imagen de 5×5 pixels de 4 bits mostrada en 7.31, realice de forma manuscrita la binarización adaptiva mediante el método de la Bradley. Use una ventana de 3×3 y replique los bordes. Explique cada paso.

Capítulo 8

El Ruido

El ruido de la imagen siempre ha sido una preocupación en la adquisición de imágenes. Él ha estado presente desde el grano en las cámaras análogas hasta hoy en los sensores de las cámaras digitales. De hecho, incluso si toma una foto en un lugar completamente oscuro o con la tapa de la lente, la imagen resultante no será totalmente negra. Puede que esté cerca, pero siempre habrá pequeñas imperfecciones: píxeles aleatorios, brillantes y descoloridos, es decir ruido. El ruido es inherente a la formación de las imágenes, por lo cual es un tema siempre vigente.

8.1. El Ruido

El ruido digital es la variación que no se corresponde con la realidad del brillo o del color en las imágenes digitales producido por el dispositivo de entrada (la cámara). El ruido puede producirse por fugas eléctricas en los circuitos que conforman el sensor (CCD o CMOS). Por simple física o por fallos de fabricación, el resultado es que un sensor elemental (píxel) termina entregando más (o menos) señal de la que generaría simplemente por la luz que incide en él, dando como resultado una imagen no uniforme.

8.2. Fuentes de Ruido

El ruido en las imágenes digitales puede provenir de varias fuentes, pero principalmente se genera en la formación y transmisión. Algunas causas son físicas, ya que están relacionadas con la naturaleza de la luz y a los elementos ópticos presentes en la formación de la imagen tales como las lentes. Otras fuentes de ruido se producen durante la captura y conversión de una señal eléctrica a datos digitales. Los sensores pueden ser inestables a la temperatura, lo que causa fluctuaciones que no corresponden necesariamente a la muestra de luz tomada por el elemento. Las principales causas de ruido en imágenes digitales surgen durante la adquisición de la imagen (digitalización) y/o la transmisión. El daño de un sensor, un contacto inadecuado o una perturbación atmosférica puede hacer que se pierdan líneas o algunos píxeles se vean modificados [40, 43].

8.2.1. Ruido Impulsivo

El ruido impulsivo altera aleatoriamente los valores de algunos píxeles. En una imagen hace que algunos elementos de la imagen se vuelvan blancos o negros sin importar su valor inicial. Es por esta razón que también es llamado ruido Sal y Pimienta (Salt & Pepper). El ruido impulsivo que puede tener muchos orígenes diferentes. Las imágenes a menudo se deterioran por el ruido impulsivo causado por errores en la captura, en transmisión, o en la localización de los bits en una memoria defectuosa. En el ruido Sal y Pimienta los píxeles ruidosos pueden tomar solo los valores máximo (255) y mínimo (0) dentro del rango dinámico, respectivamente. Se diferencia de los píxeles muertos en una pantalla en el hecho que es aleatorio en su posición, mientras que en un monitor defectuoso sucede siempre para la misma localización. La intensidad $I(x, y)$ de un porcentaje N total de los píxeles de la imagen se reemplaza de manera aleatoria por valores de 0 o 255 para así lograr el ruido Sal y Pimienta [13].

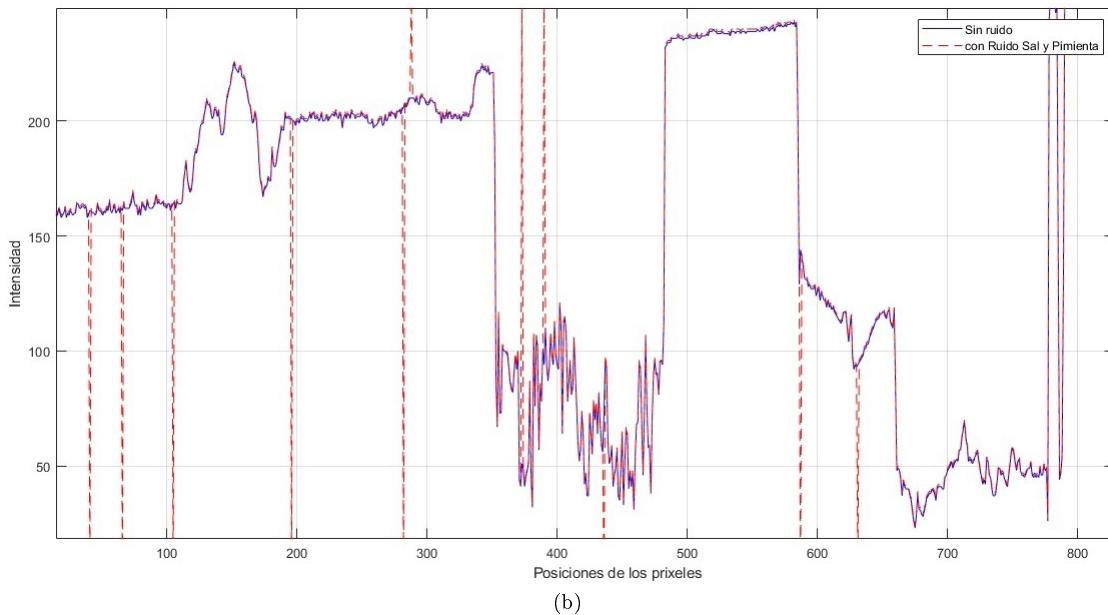
$$\delta(x, y) \begin{cases} 255 & p \\ 0 & p - 1 \end{cases} \quad (8.1)$$

$$G(x, y) = \delta(x, y)I(x, y) \quad (8.2)$$

En la Figura 8.2 se ilustra la imagen de salida con ruido impulsivo $G(x, y)$. Esta es el resultado de realizar la multiplicación de la intensidad $I(x, y)$ de cada píxel por un impulso $\delta(x, y)$ que puede tomar valores de 0 o 255 para una imagen de 8 bits. La posición de ocurrencia es también aleatoria y no todos los píxeles se ven afectados.



(a)



(b)

Figura 8.1: (a) Imagen con Ruido Sal y Pimienta y (b) corte de las misma fila sin ruido (azul) y con ruido Sal y Pimienta (punto rojo). Nótense los saltos entre 0 y 255 en las intensidades cuando hay ruido.

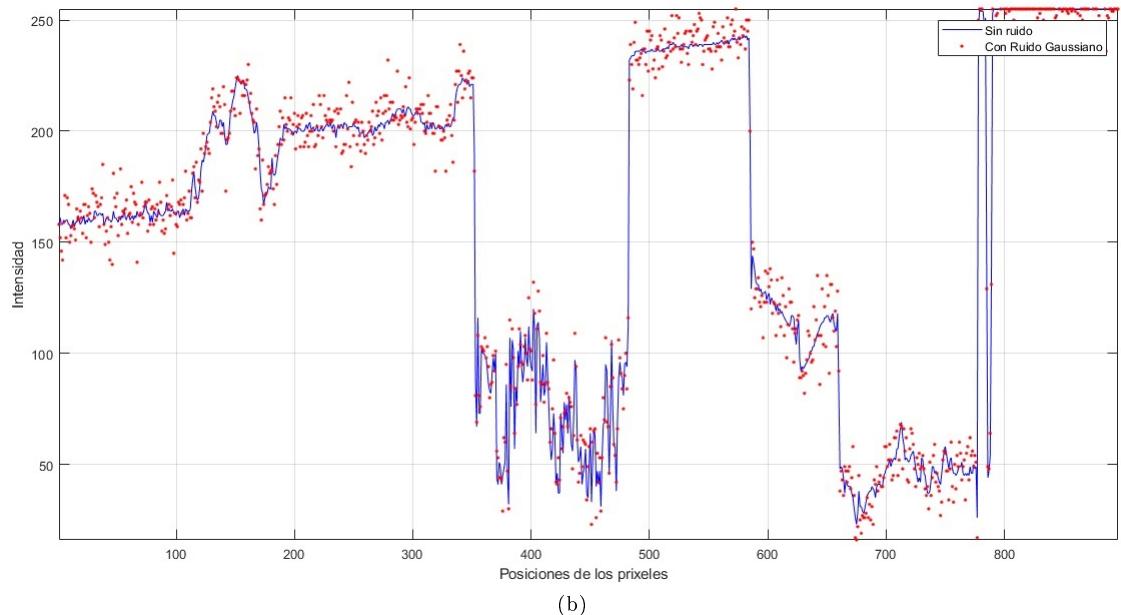
Dos aspectos a resaltar en la eliminación del ruido de sal y Pimienta son el cómo no modificar los píxeles no alterados y el cómo usar los píxeles no corruptos para estimar con precisión el píxel afectado.

8.2.2. Ruido Gaussiano

El ruido Gaussiano afecta la totalidad de los píxeles de la imagen agregando o quitando un valor de la intensidad de forma aleatoria. Todos los píxeles que componen la imagen cambian su valor a una proporción de acuerdo con la función de distribución normal o Gaussiana. En términos de ruido aditivo se podrían aplicar otro tipo de distribuciones, pero la Gaussiana se toma como el que más se aproxima, debido al teorema central de límite que indica que la suma de los diferentes ruidos tiende a aproximarse a una distribución Normal o Gaussiana.



(a)



(b)

Figura 8.2: (a) Imagen con Ruido Gaussiano y (b) corte de la misma fila sin ruido (azul) y con ruido Gaussiano (punto rojo). Nótese que las intensidades se ven afectadas de manera regular sin importar si su valor es alto o bajo. El ruido no depende de la intensidad del pixel.

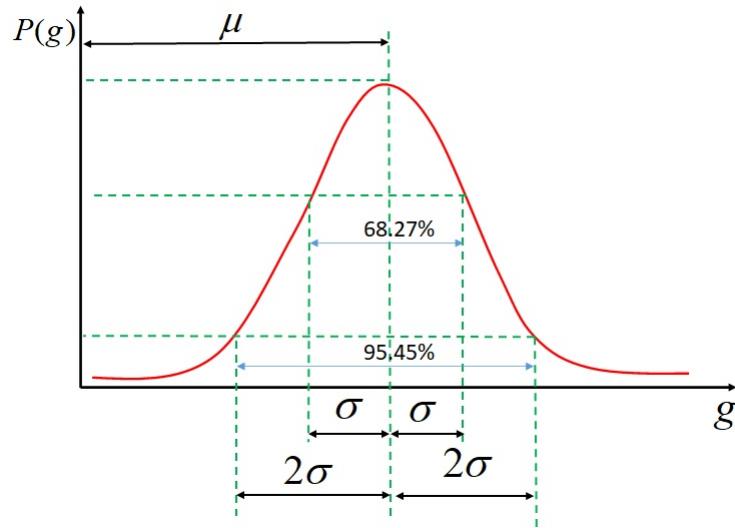


Figura 8.3: Función de distribución Gaussiana.

$$p(g) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(g-\mu)^2}{2\sigma^2}} \quad (8.3)$$

El ruido Gaussiano afecta los valores de cada píxel en la imagen. Debido a la aleatoriedad, la curva de ruido gaussiano normalizada tiene forma de campana. La función de distribución de probabilidad de ruido muestra que el 68.27% se encuentra entre $\mu - \sigma$ y $\mu + \sigma$.

Una imagen con ruido aditivo Gaussiano $G(x, y)$ se logra sumándole a la imagen sin ruido $I(x, y)$, una componente aleatoria $n(x, y)$ con función de distribución Gaussiana $p(g)$, como se muestra en (8.3) y se presenta en la Figura 8.3. Esta adición de ruido debe tener una desviación estándar sigma (σ) y promedio (μ). El modelo general de ruido aditivo Gaussiano se ilustra en (8.4) [13].

$$G(x, y) = I(x, y) + n(x, y) \quad (8.4)$$

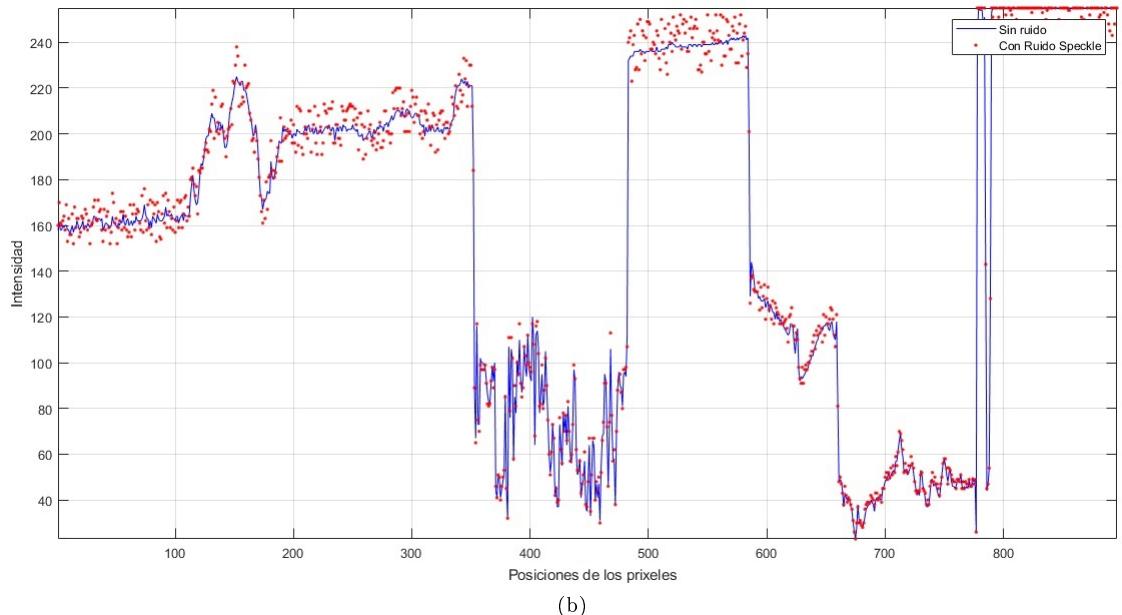
8.2.3. Ruido Speckle

El ruido speckle (moteado) es un fenómeno común en todos los sistemas de captura con fuentes coherentes como los usados para registrar imágenes tomadas con láser o ultrasonido. La causa de este tipo de ruido se debe a una interferencia aleatoria entre los retornos coherentes emitidos por los elementos dispersores presentes en la textura fotografiada, en la escala de una longitud de onda que escanea la superficie. El uso del ultrasonido para la generación de imágenes diagnósticas resulta útil gracias a su naturaleza no invasiva, su bajo costo, y su capacidad de formar imágenes en tiempo real. Sin embargo, estas imágenes se ven drásticamente afectadas por el ruido del sistema de adquisición, el ruido ambiental durante la toma, los otros órganos que generan oclusión, la grasa corporal y el movimiento respiratorio en algunas zonas corporales [13].

En general, el ruido Speckle está presente en la fotografía aérea, la médica y la satelital. El ruido de Speckle puede entenderse como una degradación granular que existe de manera inherente y afecta la calidad de la imagen. Su naturaleza es multiplicativa por lo que depende de la señal.



(a)



(b)

Figura 8.4: (a) Imagen con Ruido Speckle y (b) corte de la misma fila sin ruido (azul) y con ruido Speckle (punto rojo). Nótese que las intensidades se ven afectadas de forma irregular dependiendo de su valor. Para intensidades bajas, el ruido es poco y para intensidades altas el efecto de este es mayor. El ruido depende de la intensidad del pixel.

$$G(x, y) = I(x, y) + n(x, y)I(x, y) \quad (8.5)$$

8.3. Evaluación Referencial del Filtrado de Imágenes

El filtrado es una técnica básica dentro del procesamiento digital de imágenes, la cual permite eliminar ruido y mejorar las características visuales. Para evaluar la efectividad de un filtro, se puede hacer mediante la comparación entre la imagen original y la filtrada. En este apartado, se presentan tres métricas referenciales destacadas para

realizarlo: el Error Cuadrático Medio (MSE), la Relación Señal-Ruido de Pico (PSNR) y el Índice de Similitud Estructural (SSIM). Estas se consideran métricas de referencia completa (Full-Reference, FR), ya que requieren de la imagen original de referencia sin ruido nativo I_r y la imagen con ruido I_n agregado en algún proceso. Es de resaltar que las técnicas referencias son útiles ya que proporcionan una medida cuantitativa más exacta de las modificaciones realizadas, lo que es fundamental para el desarrollo y optimización de algoritmos del filtrado.

8.3.1. Error Cuadrático Medio (MSE)

El Error Cuadrático Medio (MSE, por sus siglas en inglés) mide la diferencia promedio al cuadrado entre los valores de los píxeles de la imagen original y la imagen filtrada. Es una medida directa de la distorsión y se utiliza comúnmente en la evaluación de la calidad de imágenes al comparar una imagen original con una imagen degradada o restaurada [13, 57].

La fórmula para calcular el MSE es:

$$\text{MSE} = \frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N (I_r(i, j) - I_n(i, j))^2$$

donde:

- M y N son las dimensiones de la imagen.
- $I_r(i, j)$ es el valor del píxel en la posición (i, j) de la imagen original.
- $I_n(i, j)$ es el valor del píxel en la posición (i, j) de la imagen con ruido.

Un valor de $\text{MSE} = 0$ indica que la imagen original y la imagen filtrada son idénticas, lo que implica una calidad perfecta de la imagen restaurada. El MSE ofrece una medida cuantitativa de la calidad, donde un valor bajo sugiere alta similitud con la imagen de referencia. Sin embargo, presenta limitaciones importantes: no está normalizado, por lo que sus valores absolutos pueden resultar difíciles de interpretar sin un contexto, y no se correlaciona necesariamente con la percepción humana de la calidad, ya que solo mide el error absoluto sin tener en cuenta aspectos estructurales o perceptuales. En muchas ocasiones, una imagen con un MSE bajo aún puede presentar artefactos visuales significativos que no son evidentes a partir de la métrica del error cuadrático. Por ello, el MSE se complementa usualmente con otras métricas como el PSNR y el Índice de Similitud Estructural (SSIM).

El Error Cuadrático Medio (MSE) se mide en las mismas unidades al cuadrado que los valores de los píxeles de la imagen. Por ejemplo, si los valores de los píxeles se encuentran en un rango de 0 a 255, como es común en imágenes de 8 bits, el MSE estará en las unidades de intensidad al cuadrado.

Ventajas:

- **Simplicidad:** Es fácil de calcular y rápido, lo que lo hace una opción popular.
- **Interpretación Directa:** Proporciona una medida directa de la magnitud del error, lo que permite evaluar de manera objetiva la cantidad de distorsión presente en una imagen.
- **Aplicabilidad Amplia:** Es una métrica estándar utilizada en muchos campos, lo que facilita la comparación entre diferentes estudios y algoritmos.

Desventajas:

- **Unidades al Cuadrado:** Al medir el error en unidades al cuadrado, el MSE puede producir valores que no son intuitivos y difíciles de interpretar en términos perceptuales, especialmente para valores de error grandes.

- **Falta de Normalización:** El MSE no está normalizado, por lo que sus valores absolutos dependen de la escala de los píxeles, lo que puede complicar la comparación entre diferentes imágenes o condiciones de evaluación.
- **No Captura la Percepción Humana:** El MSE no considera la percepción humana de la calidad, ya que mide solo el error absoluto sin tener en cuenta aspectos estructurales, texturales o visuales importantes al ojo humano.

8.3.2. Ejemplo Cálculo de MSE

Para las dos imágenes de 2×2 a 8 bits, una original de referencia I_r y la con ruido I_n , estimar el MSE a partir de sus matrices:

$$I_r = \begin{bmatrix} 100 & 150 \\ 150 & 200 \end{bmatrix} \quad I_n = \begin{bmatrix} 98 & 152 \\ 149 & 202 \end{bmatrix}$$

Solución

El *MSE* se calcula de la siguiente manera:

$$MSE = \frac{1}{2 \times 2} [(100 - 98)^2 + (150 - 152)^2 + (150 - 149)^2 + (200 - 202)^2]$$

$$MSE = \frac{1}{4} [4 + 4 + 1 + 4] = \frac{13}{4} = 3.2500$$

8.3.3. Relación Señal-Ruido de Pico (PSNR)

La Relación Señal-Ruido de Pico (PSNR, por sus siglas en inglés) es una métrica que evalúa la calidad de una imagen filtrada en comparación con la imagen original, expresando esta comparación en términos logarítmicos y en decibel (dB). A diferencia del MSE, que mide el error absoluto entre las imágenes, el PSNR traduce esa diferencia en una escala logarítmica que facilita la interpretación de la calidad percibida; un valor más alto de PSNR corresponde a una mejor calidad de imagen filtrada [13, 17]. La fórmula del PSNR es:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right)$$

donde $MAX_I = 2^{\text{bits}} - 1$ es el valor máximo posible de un píxel, comúnmente 255 para imágenes de 8 bits, y MSE representa el error cuadrático medio. Esta métrica resulta particularmente útil en aplicaciones de procesamiento de imágenes y video, ya que permite cuantificar la pérdida de calidad debido a la compresión, el filtrado o la transmisión de datos, proporcionando una manera intuitiva de evaluar la fidelidad de la imagen restaurada en comparación con su referencia ideal.

La Relación Señal-Ruido de Pico (PSNR) se mide en decibel (dB), una unidad logarítmica que expresa la relación entre el valor máximo posible de un píxel, dado el número de bits, y el error cuadrático medio (MSE). Esta refleja la calidad de la imagen de manera más intuitiva, ya que los decibel permiten comparar diferencias grandes en una escala reducida y comprensible. En términos de percepción, un PSNR más alto indica una mejor calidad de imagen, generalmente considerado aceptable a partir de los 30 dB para imágenes de 8 bits.

Ventajas:

- **Interpretación Intuitiva:** El uso de decibel facilita la comprensión de la calidad de la imagen; un PSNR más alto implica menos distorsión y mejor fidelidad.

- **Relación Directa con el MSE:** Dado que se calcula a partir del MSE, proporciona una evaluación cuantitativa del error, pero en una escala logarítmica que es más manejable.
- **Aplicación Amplia:** Es una métrica estándar y comúnmente aceptada en el procesamiento de imágenes y video, lo que facilita la comparación entre diferentes algoritmos y técnicas.

Desventajas:

- **No Captura Percepción Visual:** Al igual que el MSE, el PSNR no considera aspectos perceptuales, estructurales o visuales, por lo que puede no correlacionarse bien con la calidad percibida por los humanos.
- **Sensibilidad a Picos de Error:** Puede ser engañoso en imágenes con errores localizados (como artefactos) que impactan la percepción visual más que lo que reflejan los valores de PSNR.
- **Limitación para Altos Valores:** En algunos casos, valores muy altos de PSNR pueden no traducirse en mejoras perceptuales significativas, ya que el ojo humano no percibe las diferencias a partir de ciertos umbrales.

8.3.4. Ejemplo Cálculo de PSNR

Para las dos imágenes de 2×2 a 8 bits, una original de referencia I_r y la con ruido I_n , estimar el PSNR a partir de sus matrices:

$$I_r = \begin{bmatrix} 100 & 150 \\ 150 & 200 \end{bmatrix} \quad I_n = \begin{bmatrix} 98 & 152 \\ 149 & 202 \end{bmatrix}$$

Solución

Como ya se calculó, en el ejercicio anterior, el valor de $MSE = 3.2500$ y para una imagen de 8 bits el $MAX_I = 255$, el PSNR se calcula como:

$$PSNR = 10 \cdot \log_{10} \left(\frac{255^2}{3.2500} \right)$$

$$PSNR \approx 10 \cdot \log_{10}(19968.46) \approx 10 \cdot 4.3006 = 43.0120 \text{ dB}$$

Este resultado indica una imagen filtrada es de buena calidad.

8.3.5. Índice de Similitud Estructural (SSIM)

El Índice de Similitud Estructural (SSIM, por sus siglas en inglés) es una métrica utilizada para evaluar la similitud entre dos imágenes, enfocándose en tres aspectos clave: luminancia, contraste y estructura. Esta técnica está alineada con la percepción visual humana, lo que la hace especialmente útil para comparar imágenes de manera que refleje cómo los humanos perciben las diferencias. Este método proporciona un valor que está entre -1 y 1, donde 1 indica una similitud estructural perfecta, mientras que -1 representa una inversión completa, aunque esta situación es rara en imágenes naturales. En la práctica, los valores del SSIM se sitúan generalmente entre 0 y 1, con 0 indicando la ausencia de correlación entre las imágenes [13, 18].

El SSIM se utiliza ampliamente en aplicaciones como la compresión y restauración de imágenes, asegurando que se mantenga una alta calidad visual. En el ámbito del procesamiento de imágenes médicas, desempeña un papel relevante al garantizar que las imágenes conserven su integridad estructural, algo crítico en el diagnóstico clínico.

$$SSIM(r, n) = [l(r, n)]^\alpha \cdot [c(r, n)]^\beta \cdot [s(r, n)]^\gamma$$

donde:

- $l(r, n)$ es la comparación de luminancia, que mide la intensidad de la luz y el brillo global de las imágenes.
- $c(r, n)$ es la comparación de contraste, que evalúa la variación y diferencia de intensidades entre píxeles.
- $s(r, n)$ es la comparación de estructura que mide la correlación entre las dos imágenes utilizando la covarianza, analizando los patrones, texturas y relaciones espaciales entre píxeles.
- α, β, γ son parámetros que ajustan la importancia de cada componente. A menudo, se utiliza $\alpha = \beta = \gamma = 1$, lo que simplifica la expresión, otorgando la misma importancia a todas las variables.

Así, cada una de las componentes se define como:

$$l(r, n) = \frac{2\mu_r\mu_n + C_1}{\mu_r^2 + \mu_n^2 + C_1}, \quad c(r, n) = \frac{2\sigma_r\sigma_n + C_2}{\sigma_r^2 + \sigma_n^2 + C_2}, \quad s(r, n) = \frac{\sigma_{rn} + C_3}{\sigma_r\sigma_n + C_3}$$

donde:

- μ_r es la media de los valores de los píxeles en la imagen I_r (Referencia).
- μ_n es la media de los valores de los píxeles en la imagen I_n (Ruido).
- C_1 es una constante pequeña que evita divisiones por cero.
- σ_r es la desviación estándar de los valores de los píxeles en la imagen I_r .
- σ_n es la desviación estándar de los valores de los píxeles en la imagen I_n .
- C_2 es otra constante pequeña para evitar divisiones por cero.
- σ_{rn} es la covarianza entre las imágenes I_r y I_n .
- C_3 es una constante pequeña.

Para evitar problemas de división por cero, se introducen las constantes como funciones del rango dinámico L de los valores de los píxeles (por ejemplo, $L = 255$ para imágenes de 8 bits):

$$C_1 = (k_1 \cdot L)^2, \quad C_2 = (k_2 \cdot L)^2, \quad C_3 = \frac{C_2}{2}$$

Donde k_1 y k_2 son constantes muy pequeñas, típicamente $k_1 = 0.01$ y $k_2 = 0.03$. Para $L = 255$, se tiene:

$$C_1 = (0.01 \cdot 255)^2 = 6.5025, \quad C_2 = (0.03 \cdot 255)^2 = 58.5225, \quad C_3 = \frac{C_2}{2} = 29.2612.$$

Ventajas:

- **Alineación con la Percepción Humana:** Está diseñado para imitar la forma en que el sistema visual humano percibe las diferencias entre imágenes, proporcionando una evaluación de calidad más alineada con la percepción visual que métricas tradicionales como el MSE o PSNR.

- **Sensibilidad a Cambios Estructurales:** Es particularmente efectivo para detectar cambios en la estructura, luminancia y contraste de las imágenes, lo que lo hace ideal para evaluar la calidad en aplicaciones como compresión y restauración de imágenes.
- **Aplicación Amplia:** Se utiliza en una variedad de campos, desde la compresión de video y transmisión en streaming hasta el procesamiento de imágenes médicas, asegurando que se mantenga la integridad visual y estructural en cada aplicación.
- **Interpretación Intuitiva:** Los valores del SSIM se sitúan típicamente entre 0 y 1, donde valores más cercanos a 1 indican una alta similitud, lo cual es fácil de interpretar y comparar.

Desventajas:

- **Limitado a Comparaciones de Imágenes del Mismo Tamaño:** El SSIM requiere que las imágenes a comparar tengan el mismo tamaño y alineación, lo cual puede limitar su uso en ciertas aplicaciones donde las imágenes están desincronizadas o tienen resoluciones diferentes.
- **No Captura Todos los Tipos de Distorsiones:** Aunque es sensible a cambios estructurales, puede no detectar ciertas distorsiones perceptualmente significativas, como ligeros cambios de color o artefactos que no afectan directamente la estructura.
- **Sensibilidad a Parámetros de Implementación:** El resultado del SSIM puede variar dependiendo de la implementación, como el tamaño de las ventanas de comparación o los parámetros de normalización, lo que puede afectar la consistencia de los resultados entre diferentes estudios.
- **Computacionalmente Más Complejo:** Comparado con métricas más simples como el MSE, el cálculo del SSIM es más complejo y puede requerir un mayor tiempo de procesamiento, especialmente en imágenes de alta resolución o en aplicaciones en tiempo real.

8.3.6. Ejemplo Cálculo de SSMI

Para las dos imágenes de 2×2 a 8 bits, una original de referencia I_r y la con ruido I_n , estimar el SSMI a partir de sus matrices:

$$I_r = \begin{bmatrix} 100 & 150 \\ 150 & 200 \end{bmatrix} \quad I_n = \begin{bmatrix} 98 & 152 \\ 149 & 202 \end{bmatrix}$$

Solución

En el contexto de las imágenes I_r e I_n de tamaño $M \times N$, el promedio de los valores de los píxeles de una imagen se puede calcular utilizando:

$$u = \frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N I(i, j)$$

Así, para u_r :

$$u_r = \frac{1}{2 \times 2} (I_r(1,1) + I_r(1,2) + I_r(2,1) + I_r(2,2)) = \frac{1}{4} (100 + 150 + 150 + 200) = \frac{600}{4} = 150$$

Y para u_n :

$$u_n = \frac{1}{2 \times 2} (I_n(1,1) + I_n(1,2) + I_n(2,1) + I_n(2,2)) = \frac{1}{4} (98 + 152 + 149 + 202) = \frac{601}{4} \approx 150.25$$

En el contexto de las imágenes I_r e I_n de tamaño $M \times N$, la desviación estándar se puede calcular utilizando la siguiente fórmula:

$$\sigma = \sqrt{\frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N [I(i,j) - u]^2}$$

La desviación estándar para la imagen de referencia I_r se expresa como:

$$\sigma_r = \sqrt{\frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N (I_r(i,j) - u_r)^2}$$

Sustituyendo los valores:

$$\begin{aligned} \sigma_r &= \sqrt{\frac{1}{2 \times 2} [(100 - 150)^2 + (150 - 150)^2 + (150 - 150)^2 + (200 - 150)^2]} = \sqrt{\frac{1}{4} [(-50)^2 + 0 + 0 + (50)^2]} \\ &= \sqrt{\frac{1}{4} (2500 + 0 + 0 + 2500)} = \sqrt{\frac{5000}{4}} = \sqrt{1250} \approx 35.36 \end{aligned}$$

La desviación estándar para la imagen a evaluar I_n se expresa como:

$$\sigma_n = \sqrt{\frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N (I_n(i,j) - u_n)^2}$$

Sustituyendo los valores:

$$\begin{aligned} \sigma_n &= \sqrt{\frac{1}{2 \times 2} [(98 - 150.25)^2 + (152 - 150.25)^2 + (149 - 150.25)^2 + (202 - 150.25)^2]} \\ &= \sqrt{\frac{1}{4} [(-52.25)^2 + (1.75)^2 + (-1.25)^2 + (51.75)^2]} = \sqrt{\frac{1}{4} [2731.5625 + 3.0625 + 1.5625 + 2677.5625]} \\ &= \sqrt{\frac{5413.75}{4}} = \sqrt{1353.4375} \approx 36.79 \end{aligned}$$

La covarianza σ_{rn} se calcula como:

$$\sigma_{rn} = \frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N (I_r(i,j) - u_r)(I_n(i,j) - u_n)$$

Sustituyendo los valores:

$$\begin{aligned}\sigma_{rn} &= \frac{1}{2 \times 2} [(100 - 150)(98 - 150.25) + (150 - 150)(152 - 150.25) + (150 - 150)(149 - 150.25) + (200 - 150)(202 - 150.25)] \\ &= \frac{1}{4} [(-50)(-52.25) + 0 + 0 + (50)(51.75)] \\ &= \frac{1}{4} [2612.5 + 0 + 0 + 2587.5] = \frac{1}{4} [5200] = 1300\end{aligned}$$

Definiendo C_1, C_2 , y C_3 para $L = 255$:

$$k_1 = 0.01, \quad k_2 = 0.03, \quad L = 255$$

$$C_1 = (k_1 \cdot L)^2 = (0.01 \cdot 255)^2 = 6.5025$$

$$C_2 = (k_2 \cdot L)^2 = (0.03 \cdot 255)^2 = 58.5225$$

$$C_3 = \frac{\sigma_2}{2} = \frac{58.5225}{2} = 29.2612$$

Calculando las Componentes del SSIM.

$$l(r, n) = \frac{2\mu_r\mu_n + C_1}{\mu_r^2 + \mu_n^2 + C_1}, \quad c(r, n) = \frac{2\sigma_r\sigma_n + C_2}{\sigma_r^2 + \sigma_n^2 + C_2}, \quad s(r, n) = \frac{\sigma_{rn} + C_3}{\sigma_r\sigma_n + C_3}$$

Sustituyendo los valores:

$$l(r, n) = \frac{2 \times 150 \times 150.25 + 6.5025}{150^2 + (150.25)^2 + 6.5025} \approx 0.9998$$

$$c(r, n) = \frac{2 \times 35.36 \cdot 36.79 + 58.5225}{35.36^2 + 36.79^2 + 58.5225} \approx 0.9853$$

$$s(r, n) = \frac{1300 + 29.26125}{35.36 \times 36.79 + 29.26125} \approx 0.9931$$

Finalmente, el SSIM total se calcula como:

$$SSIM(r, n) = l(r, n) \cdot c(r, n) \cdot s(r, n) \approx 0.9998 \times 0.9853 \times 0.9931 \approx 0.9783$$

Este valor indica que las dos imágenes tienen una alta similitud estructural.

La función `ssim` de MATLAB calcula el índice de similitud estructural (SSIM) usando un enfoque local, en el que la imagen se divide en pequeñas ventanas deslizantes (generalmente de 11×11 píxeles), y se evalúa el SSIM de forma individual. A diferencia del cálculo manual global que se muestra aquí, el usado por MATLAB permite captar variaciones locales en luminancia, contraste y estructura, lo que refleja mejor cómo el ojo humano percibe la similitud visual. Además, aplican un suavizado gaussiano dentro de cada ventana, dando mayor peso a los píxeles cercanos al centro, lo que ayuda a que la medida sea más ajustada en áreas de la imagen con cambios sutiles. Este enfoque local y suavizado da como resultado un valor SSIM final que suele diferir del cálculo manual, proporcionando una evaluación de calidad visual más detallada y realista.

8.4. Funciones

Las siguientes funciones tienen relación con el tema abordado en este capítulo:

1. `imnoise(I, 'tipo', parámetros)`: agrega ruido a una imagen `I`, simulando distintos tipos de perturbaciones comunes en imágenes digitales. El segundo argumento especifica el tipo de ruido, como '`salt & pepper`', '`gaussian`', '`poisson`' o '`speckle`'. Según el tipo de ruido, se pueden agregar parámetros opcionales como la densidad de impulsos o la varianza. Esta función es útil para probar la robustez de algoritmos ante degradaciones.
2. `psnr(A, B)`: calcula la relación señal-ruido pico (Peak Signal-to-Noise Ratio) entre la imagen original `A` y una imagen distorsionada `B`. El resultado se expresa en decibelios (dB), y un valor más alto indica mayor similitud. Es una métrica ampliamente utilizada para evaluar la calidad de compresión, restauración o transmisión de imágenes.
3. `ssim(A, B)`: evalúa la similitud estructural (Structural Similarity Index) entre dos imágenes `A` y `B`, considerando luminancia, contraste y estructura local. El índice SSIM varía entre 0 y 1, donde 1 representa similitud perfecta. A diferencia de PSNR o MSE, tiene una mejor correlación con la percepción visual humana.
4. `immse(A, B)`: calcula el error cuadrático medio (Mean Squared Error) entre las imágenes `A` y `B`. Esta métrica representa el promedio de las diferencias al cuadrado entre los valores de píxel correspondientes, y se utiliza para cuantificar la distorsión global entre dos imágenes del mismo tamaño.

8.5. Preguntas

8.5.1. Preguntas sobre hechos del texto

1. ¿Cuáles son las tres métricas referenciales destacadas para evaluar la efectividad del filtrado de imágenes mencionadas en el documento?
2. ¿En qué unidades se mide el Error Cuadrático Medio (MSE)?
3. ¿Qué tres aspectos clave evalúa el Índice de Similitud Estructural (SSIM)?
4. ¿Cuál es el rango de valores que puede tomar el SSIM y qué indican estos valores?
5. ¿Cuál es el valor típico de PSNR considerado aceptable para imágenes de 8 bits?

8.5.2. Preguntas de análisis y comprensión sobre el texto

1. ¿Por qué el MSE, a pesar de ser una medida directa de distorsión, tiene limitaciones en la evaluación de la calidad de imágenes? Explique.
2. Compare las ventajas y desventajas del PSNR frente al MSE en la evaluación de la calidad de imágenes filtradas.

3. ¿Cómo se relaciona el SSIM con la percepción visual humana y por qué esto es importante en el procesamiento de imágenes?
4. Analice por qué el SSIM podría ser preferible al MSE o PSNR en aplicaciones de procesamiento de imágenes médicas.
5. ¿Cómo afecta la escala logarítmica del PSNR a la interpretación de la calidad de la imagen en comparación con el MSE?

8.5.3. Problemas Numéricos sobre Ruido en Imágenes

1. Calcule el Error Cuadrático Medio (MSE), el PSNR y el SSIM para las imágenes I_r de referencia e I_n con ruido de 2×2 píxeles:

$$I_r = \begin{bmatrix} 100 & 150 \\ 200 & 250 \end{bmatrix} \quad I_{nr} = \begin{bmatrix} 105 & 148 \\ 202 & 245 \end{bmatrix}$$

2. Dadas dos imágenes de 8 bits con un MSE calculado de 100, determine el valor de PSNR.
3. Calcule los componentes de luminancia l y contraste c del SSIM para las imágenes I_r de referencia e I_n con ruido de 2×2 píxeles:

$$I_r = \begin{bmatrix} 50 & 100 \\ 150 & 200 \end{bmatrix} \quad I_n = \begin{bmatrix} 55 & 98 \\ 152 & 195 \end{bmatrix}$$

Utilice $C_1 = 6.5025$ y $C_2 = 58.5225$.

4. Una imagen de 512×512 píxeles tiene un valor máximo de píxel de 255. Si el PSNR entre esta y una versión comprimida es de $30dB$, ¿cuál es el valor del MSE?
5. Calcule el SSIM total para dos imágenes, dado que:

$$l(r, n) = 0.98, c(r, n) = 0.95, s(r, n) = 0.97$$

Asuma que $\alpha = \beta = \gamma = 1$ en la fórmula del SSIM.

Capítulo 9

Filtros Espaciales

9.1. Filtrado Espacial de Imágenes

Es una técnica esencial en el procesamiento de imágenes que emplea operaciones con ventanas para modificar las características locales. Esto permite realizar la eliminación de ruido, mejora de calidad, enfatización o atenuación de aspectos específicos, detección de bordes y suavizado. Este método opera sobre múltiples píxeles simultáneamente, utilizando ventanas de tamaño variable, recorriendo la imagen píxel por píxel, realizando operaciones aritméticas o estadísticas con los vecinos [5, 13].

9.2. Operaciones Espaciales

Para efectos de esta sección, se aclaran los siguientes conceptos:

Ventana: se refiere al área local de la imagen que se selecciona y examina para aplicar una operación. Esta se desplaza por toda la imagen, abarcando diferentes regiones de píxeles en cada paso.

Kernel: es la matriz de valores que tiene los coeficientes o pesos que se aplican a las intensidades píxeles dentro de cada ventana para generar el valor filtrado. En resumen, la ventana delimita el área que se procesa, y el kernel especifica cómo se transforma la información contenida en esa área.

9.2.1. Correlación en Dos Dimensiones

La correlación entre una imagen $I(x, y)$ y una ventana $k(x, y)$ en dos dimensiones se define como:

$$I'(x, y) = \sum_{i=-m}^m \sum_{j=-n}^n I(x+i, y+j) \cdot k(i, j)$$

que también se puede escribir como:

$$I' = I \star k$$

En esta definición:

- $I(x+i, y+j)$ representa los valores de la imagen I desplazada por i y j .
- $k(i, j)$ es la ventana utilizada para calcular la correlación.

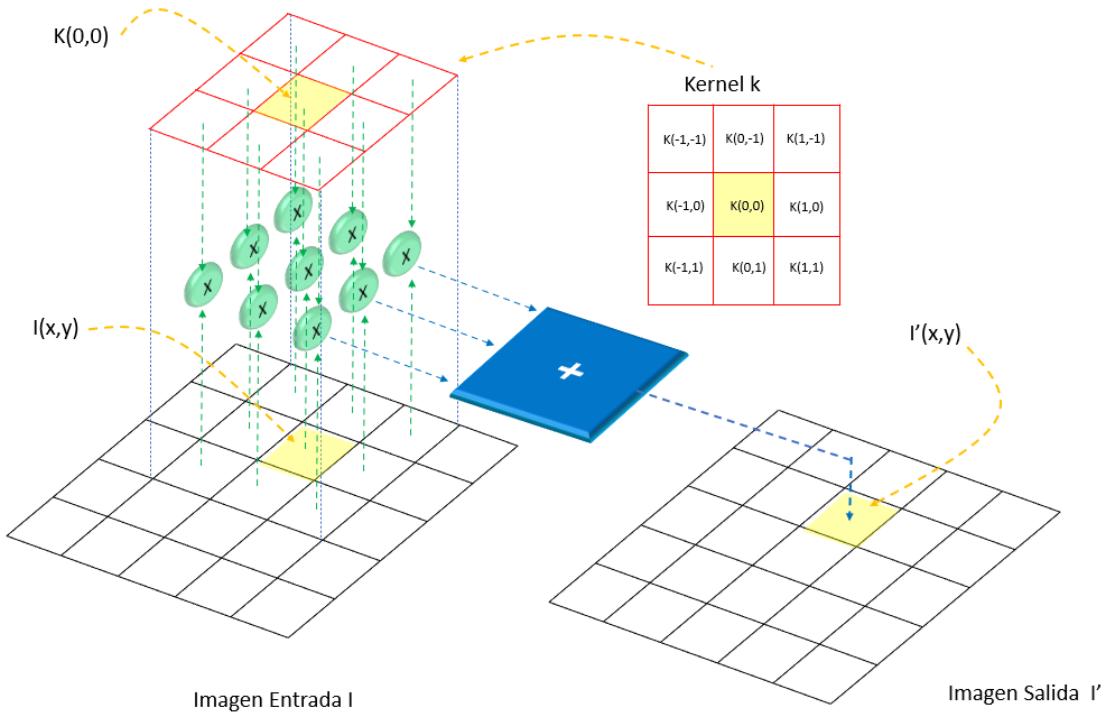


Figura 9.1: Principio conceptual de la correlación en imágenes: En el proceso consiste en desplazar el kernel K sobre la imagen $I(x, y)$. Para cada posición del kernel, se realizan multiplicaciones elemento a elemento entre los valores del kernel y los correspondientes de la imagen bajo el kernel. Posteriormente, se suman todos los productos obtenidos. El valor resultante se asigna a la posición $I'(x, y)$ en la imagen de salida. Esto se repite a medida que el kernel se mueve por toda la imagen, produciendo una nueva filtrada.

- Los índices i y j recorren un rango que va desde $-m$ hasta m y desde $-n$ hasta n , respectivamente.

De manera alternativa, la correlación en 2 dimensiones puede notarse usando el producto de matrices como $I' = I \star k$, donde el operador \star indica la correlación. En el procesamiento de imágenes, esta operación, implica recorrer la imagen de entrada $I(x, y)$ aplicando un kernel $k(x, y)$ en cada píxel.

La Figura 9.1 muestra de manera conceptual el proceso de correlación, que también es válido para correlación si se rota el kernel como se explica más adelante.

9.2.2. Convolución en Dos Dimensiones

La convolución entre una imagen $I(x, y)$ y una ventana $k(x, y)$ en dos dimensiones se define como:

$$I'(x, y) = \sum_{i=-m}^m \sum_{j=-n}^n I(x - i, y - j) \cdot k(i, j)$$

En esta definición:

- $I(x - i, y - j)$ representa los valores de la imagen I desplazada por $-i$ y $-j$.
- $k(i, j)$ es la ventana utilizada para calcular la convolución.
- Los índices i y j recorren un rango que va desde $-m$ hasta m y desde $-n$ hasta n , respectivamente.

Alternativamente, la convolución también puede resolverse con la rotación del kernel k , lo cual se logra haciendo el

cambio de los signos y tratándolo similarmente como una correlación.

$$I'(x, y) = \sum_{i=-m}^m \sum_{j=-n}^n I(x+i, y+j) \cdot k(-i, -j)$$

que también se puede escribir como:

$$I' = I * k$$

En esta definición:

- $I(x+i, y+j)$ representa los valores de la imagen I desplazada por i y j .
- $k(-i, -j)$ es la ventana utilizada para calcular la correlación, lo cual implica una rotación de 180 grados.
- Los índices i y j recorren un rango que va de $-m$ a m y de $-n$ a n dentro de la ventana, respectivamente.

De manera similar a como sucede con la correlación, la convolución en 2 dimensiones puede representarse usando la notación de producto de matrices como $I' = I * k$, donde el operador $*$ denota la convolución.

La convolución se puede resolver como una correlación si se rota el kernel 180 grados. Esto se debe a que ambas son operaciones similares, pero con la diferencia fundamental en la orientación del kernel. En la convolución, el kernel se rota antes de aplicarse a la imagen a través de la correlación, mientras que en la correlación el kernel se aplica directamente en su orientación original.

9.2.3. Ejemplo: evaluación de la Correlación y la Convolución

Calcular la correlación ($I' = I * k$) y la convolución ($I' = I * k$) entre la imagen I y el kernel k para obtener I' para el píxel $I(2, 2)$ usando las definiciones. Los valores de las matrices son los siguientes:

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad k = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Solución

1. Para la correlación:

$$\begin{aligned} I'(2, 2) &= \sum_{i=-1}^1 \sum_{j=-1}^1 I(2+i, 2+j) \cdot k(i, j) \\ &= I(1, 1) \cdot k(-1, -1) + I(1, 2) \cdot k(-1, 0) + I(1, 3) \cdot k(-1, 1) + \dots \\ &\quad + I(2, 1) \cdot k(0, -1) + I(2, 2) \cdot k(0, 0) + I(2, 3) \cdot k(0, 1) + \dots \\ &\quad + I(3, 1) \cdot k(1, -1) + I(3, 2) \cdot k(1, 0) + I(3, 3) \cdot k(1, 1) \end{aligned}$$

Sustituyendo los valores de I y k para evaluar la correlación en el píxel central $I'(2, 2)$ es:

$$I'(2, 2) = 1a + 2b + 3c + 4d + 5e + 5f + 7g + 8h + 9i$$

2. Para la convolución:

$$\begin{aligned}
I'(2, 2) &= \sum_{i=-1}^1 \sum_{j=-1}^1 I(2-i, 2-j) \cdot k(i, j) \\
&= I(3, 3) \cdot k(-1, -1) + I(3, 2) \cdot k(-1, 0) + I(3, 1) \cdot k(-1, 1) + \dots \\
&\quad + I(2, 3) \cdot k(0, -1) + I(2, 2) \cdot k(0, 0) + I(2, 1) \cdot k(0, 1) + \dots \\
&\quad + I(1, 3) \cdot k(1, -1) + I(1, 2) \cdot k(1, 0) + I(1, 1) \cdot k(1, 1)
\end{aligned}$$

Sustituyendo los valores de I y k para evaluar la convolución en el píxel central $I'(2, 2)$ es:

$$I'(2, 2) = 9a + 8b + 7c + 6d + 5e + 4f + 3g + 2h + 1i$$

De forma alternativa, la convolución puede calcularse como:

$$\begin{aligned}
I'(2, 2) &= \sum_{i=-1}^1 \sum_{j=-1}^1 I(2+i, 2+j) \cdot k(-i, -j) \\
&= I(1, 1) \cdot k(1, 1) + I(1, 2) \cdot k(1, 0) + I(1, 3) \cdot k(1, 1) + \dots \\
&\quad + I(2, 1) \cdot k(0, 1) + I(2, 2) \cdot k(0, 0) + I(2, 3) \cdot k(0, -1) + \dots \\
&\quad + I(3, 1) \cdot k(-1, 1) + I(3, 2) \cdot k(-1, 0) + I(3, 3) \cdot k(-1, -1)
\end{aligned}$$

Sustituyendo los valores de I y k para evaluar la convolución en el píxel central $I'(2, 2)$ también es:

$$I'(2, 2) = 1i + 2h + 3g + 4f + 5e + 6d + 7c + 8b + 9a$$

9.3. Equivalencia entre Correlación y Convención

Los cálculos de correlación y convolución son equivalentes si el kernel $k(i, j)$ es simétrico, es decir, si:

$$k(i, j) = k(-i, -j)$$

En ese caso, la operación de convolución se convierte en una correlación, y viceversa. Matemáticamente, la convolución de una imagen de entrada $I(x, y)$ con un kernel $k(x, y)$ se define como:

$$I'(x, y) = \sum_{i=-m}^m \sum_{j=-n}^n I(x-i, y-j) \cdot k(i, j)$$

Si el kernel $k(i, j)$ es simétrico, entonces se puede escribir:

$$k(i, j) = k(-i, -j)$$

Sustituyendo en la definición de la convolución, se llega a:

$$I'(x, y) = \sum_{i=-m}^m \sum_{j=-n}^n I(x-i, y-j) \cdot k(-i, -j)$$

Lo cual es equivalente a la correlación:

$$I'(x, y) = \sum_{i=-m}^m \sum_{j=-n}^n I(x+i, y+j) \cdot k(i, j)$$

Convolución vs Correlación

Es más común hablar de convolución que de correlación en el procesamiento de imágenes y en los campos relacionados con el análisis de señales por varias razones:

- **Propiedades matemáticas:** La convolución tiene propiedades matemáticas que son muy útiles, como la comutatividad, la asociatividad y la distributividad. Estas facilitan la manipulación y el análisis de señales e imágenes.
- **Transformada de Fourier:** La convolución en el dominio espacial se convierte en una multiplicación en el dominio de la frecuencia (según la transformada de Fourier). Esto simplifica enormemente el proceso de filtrado y análisis de señales.

9.3.1. Ejemplo: evaluación de la Convolución usando la Correlación

Calcular la convolución ($I' = I * k$) entre la imagen I y el kernel k para obtener $I'(2, 2)$ usando la expresión de correlación.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad k = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

9.4. Solución

Para hacerlo, se debe rotar el kernel k 180 grados, lo cual se logra al invertir horizontalmente y luego verticalmente sus valores, así:

$$k' = \begin{bmatrix} i & h & g \\ f & e & d \\ c & b & a \end{bmatrix}$$

Para la correlación con el kernel k' rotado 180 grados sería:

$$I'(2, 2) = \sum_{i=-1}^1 \sum_{j=-1}^1 I(2+i, 2+j) \cdot k'(i, j)$$

$$\begin{aligned} &= I(1, 1) \cdot k'(-1, -1) + I(1, 2) \cdot k'(-1, 0) + I(1, 3) \cdot k'(-1, 1) + \dots \\ &\quad + I(2, 1) \cdot k'(0, -1) + I(2, 2) \cdot k'(0, 0) + I(2, 3) \cdot k'(0, 1) + \dots \end{aligned}$$

$$+I(3,1) \cdot k'(1,-1) + I(3,2) \cdot k'(1,0) + I(3,3) \cdot k'(1,1)$$

Por lo tanto, el resultado de la convolución para el píxel central $I'(2,2)$ es:

$$I'(2,2) = 1i + 2h + 3g + 4f + 5e + 6d + 7c + 8b + 9a$$

Nótese que los resultados son iguales a los calculados anteriormente correspondientes a la convolución.

9.5. Tipos de Relleno

En el procesamiento de imágenes, la convolución y la correlación en 2D son operaciones comunes que se realizan en matrices de datos. Sin embargo, al aplicar estas operaciones en los bordes de una imagen, es necesario realizar un relleno (padding) para evitar la pérdida de información o la reducción del tamaño la imagen de salida. Este proceso añade valores adicionales alrededor de la imagen original, de modo que la ventana de convolución o correlación pueda desplazarse por todos los píxeles sin problemas de pérdidas en los bordes.

Considere una imagen de 3×3 :

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

9.5.1. Relleno de Constante (Constant Padding)

Esta rellena los bordes de la imagen con un valor constante. La estrategia es simple y flexible, ya que permite controlar fácilmente el valor del borde. Sin embargo, una desventaja es que puede introducir bordes artificiales que no representan los datos originales de la imagen.

Ejemplo:

$$\begin{bmatrix} k & k & k & k & k & k & k \\ k & k & k & k & k & k & k \\ k & k & a & b & c & k & k \\ k & k & d & e & f & k & k \\ k & k & g & h & i & k & k \\ k & k & k & k & k & k & k \\ k & k & k & k & k & k & k \end{bmatrix}$$

9.5.2. Relleno Cero (Zero Padding)

El relleno cero (Zero Padding) consiste en agregar filas y columnas de ceros alrededor de la imagen original. Esta estrategia es simple de implementar y no introduce nuevos valores que puedan alterar los resultados de la operación. Sin embargo, una desventaja importante es que puede generar bordes artificiales de cero que no reflejan los datos originales de la imagen, lo que puede repercutir en los cálculos. Es un caso espacial del Relleno de Constante.

Ejemplo:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a & b & c & 0 & 0 \\ 0 & 0 & d & e & f & 0 & 0 \\ 0 & 0 & g & h & i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

9.5.3. Relleno Replicado (Replicate Padding)

Este tipo de relleno implica repetir el mismo valor del borde más cercano alrededor de la imagen original. Esta estrategia es sencilla de implementar y asegura que los valores agregados estén dentro del rango de los datos originales. Sin embargo, una desventaja es que puede causar duplicación poco natural en los bordes cuando se aleja, lo que puede introducir artefactos no deseados en la imagen.

Ejemplo:

$$\begin{bmatrix} a & a & a & b & c & c & c \\ a & a & a & b & c & c & c \\ a & a & a & b & c & c & c \\ d & d & d & e & f & f & f \\ g & g & g & h & i & i & i \\ g & g & g & h & i & i & i \\ g & g & g & h & i & i & i \end{bmatrix}$$

9.5.4. Relleno Simétrico (Symmetric Padding)

Esta estrategia mantiene la simetría de la imagen en los bordes a manera de espejo. Sin embargo, una desventaja es que puede duplicar valores de manera poco realista, lo que puede afectar la calidad de la imagen procesada. El relleno simétrico puede producir resultados más suaves, pero también puede introducir valores duplicados que no existen en la imagen original.

Ejemplo:

$$\begin{bmatrix} e & d & d & e & f & f & e \\ b & a & a & b & c & c & b \\ b & a & a & b & c & c & b \\ e & d & d & e & f & f & e \\ h & g & g & h & i & i & h \\ h & g & g & h & i & i & h \\ e & d & d & e & f & f & e \end{bmatrix}$$

9.5.5. Relleno Cíclico (Circular Padding)

Este implica llenar los bordes de la imagen con valores del lado opuesto de la imagen. Esta aproximación es útil para señales periódicas, ya que mantiene la continuidad de la señal en los bordes de la imagen. Sin embargo, una desventaja es que no siempre es adecuado para datos no periódicos, ya que puede introducir artefactos en la imagen si la información no siguen un patrón periódico claro.

Ejemplo:

$$\begin{bmatrix} i & h & g & h & i & h & g \\ f & e & d & e & f & e & d \\ c & b & a & b & c & b & a \\ i & h & g & h & i & h & g \\ f & e & d & e & f & e & d \\ c & b & a & b & c & b & a \\ i & h & g & h & i & h & g \end{bmatrix}$$

9.6. Filtros de Suavizado (Pasa-Bajo)

9.6.1. Filtro Aritmético

El filtro promedio aritmético es uno de los más simples que se utilizan en el procesamiento de imágenes. Su principio de funcionamiento consiste en recorrer toda la imagen con un ventana rectangular de tamaño $p \times q$, centrada en el punto (x, y) . Para cada posición de la imagen, se calcula el promedio aritmético de la intensidad de los píxeles que coinciden con el centro de la máscara, lo cual se logra sumando todos los valores y dividiéndolos por el número total de elementos $(p \times q)$ [5, 13]. El nuevo valor de la intensidad del píxel central $I'(x, y)$ se obtiene mediante la expresión (9.1):

$$I'(x, y) = \frac{1}{p \times q} \sum_{i=-m}^m \sum_{j=-n}^n I(x+i, y+j) \quad (9.1)$$

Alternativamente, esta operación también se puede expresar como una convolución:

$$I'(x, y) = I(x, y) * h(x, y)$$

donde $h(x, y)$ se define como una matriz de unos de tamaño $p \times q$:

$$h(x, y) = \frac{1}{p \times q} \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 1 & \cdots & \cdots & \cdots & 1 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & \cdots & \cdots & \cdots & 1 \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \text{ donde } p = 2m + 1 \text{ y } q = 2n + 1$$

Este filtro es adecuado para reducir el ruido de alta frecuencia en la imagen, pero también puede causar una pérdida excesiva de detalles finos. Las coordenadas de los píxeles de origen son fijas con respecto a la posición actual (x, y) y, por lo general, forman una región contigua, como se ilustra en la Figura 9.2.

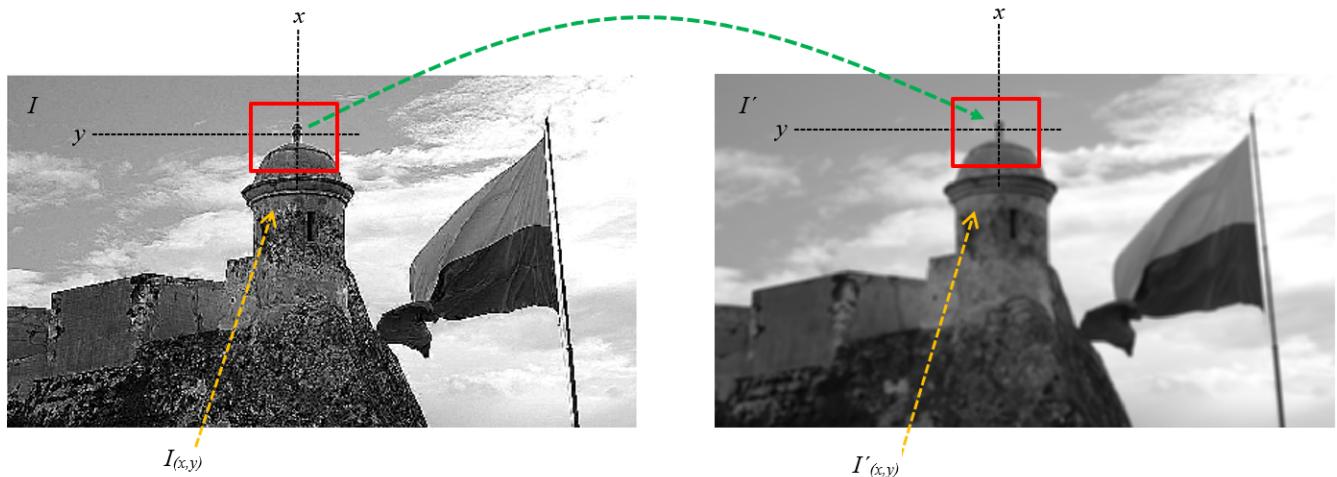


Figura 9.2: Funcionamiento principal del filtro. La imagen I se convierte en I' una vez se ha aplicado el filtro promedio aritmético.

El *tamaño* de la región del filtro es un parámetro importante del mismo, ya que especifica cuántos píxeles originales contribuyen a cada valor de la intensidad del pixel resultante y por lo tanto determina la extensión espacial (soporte).

Por ejemplo, el filtro de suavizado que utiliza una región 3×3 , tiene menor influencia sobre el promedio que otro con ventana de vecindad mayor. El filtro promedio aritmético considera todos los píxeles dentro de una ventana de tamaño fijo, sin tener en cuenta la distancia a la que se encuentran del píxel central. Esto significa que el valor de cada píxel dentro de la ventana se pondera igualmente al calcular el promedio. El hecho de no tener en cuenta la distancia resulta algunas veces problemático, ya que los más alejados del centro de la ventana pueden tener valores significativamente diferentes que no reflejan correctamente las características locales de la imagen, lo que introduce artefactos en la imagen procesada.

9.6.2. Filtros de Orden

Los filtros de orden operan mediante la ordenación de las intensidades de los píxeles dentro de una ventana de tamaño fijo y seleccionando un valor específico de la secuencia. Dependiendo del valor escogido, el filtro puede realizar diferentes funciones tales como el de mínimo, el de máximo y la de mediana.

Para un píxel en la posición (x, y) de una imagen I , el valor filtrado $I'(x, y)$ se obtiene mediante una ventana R de tamaño $p \times q$ centrada en (x, y) [2, 5, 13, 55]. Los valores de los píxeles dentro esta se ordenan, y se selecciona uno de ellos según el criterio del filtro dado usando la expresión (9.2):

$$I'(x, y) = \text{orden}(\{I(x + i, y + j) \mid (i, j) \in R\}) \quad (9.2)$$

En general, R es la región definida por la ventana del filtro la cual es usualmente impar $p \times q$, con una secuencia ordenada de valores $p = (p_0, p_1, \dots, p_{K-1}, p_K, p_{K+1}, \dots, p_{2K})$, donde $K = \frac{pq+1}{2}$. Así, el valor mínimo es p_0 , el central p_K y el máximo p_{2K} .

Filtro Mínimo

El filtro de mínimo selecciona el valor más pequeño dentro de la ventana. Esto significa que para cada píxel de la imagen, el valor filtrado es el valor mínimo entre todos los píxeles dentro de la ventana. Consiste en recorrer la imagen con una ventana de filtrado R y reemplazar el valor del píxel central de la ventana por el valor mínimo de todos los píxeles dentro de la ventana usando la definición presente en (9.3).

$$I'(x, y) = \min \{I(x + i, y + j) \mid (i, j) \in R\} \quad (9.3)$$

R es la región definida por la ventana del filtro la cual es usualmente impar $p \times q$, con una secuencia ordenada de valores $p = (p_0, p_1, \dots, p_{K-1}, p_K, p_{K+1}, \dots, p_{2K})$, donde $K = \frac{pq+1}{2}$. Así, el valor mínimo es p_0

Entre sus ventajas, se destaca su eficacia para eliminar el ruido positivo (puntos brillantes) y la capacidad para preservar los bordes en las áreas oscuras de la imagen. Sin embargo, también tiene desventajas, como el aumento del ruido negativo (puntos oscuros) y la pérdida de detalles en áreas brillantes. La Figura 9.3 muestra el efecto de usar el filtro máximo sobre una imagen con Sal y Primienta. Esto, como era de esperarse, enfatiza los puntos negros de la pimienta.

Filtro Máximo

El filtro de máximo selecciona el valor más grande dentro de la ventana. Esto significa que para cada píxel de la imagen, el valor filtrado es el valor máximo entre todos los píxeles dentro de la ventana. De manera similar al mínimo, consiste en recorrer la imagen con una ventana de filtrado R y reemplazar el valor del píxel central de la ventana por el valor máximo de todos los píxeles dentro de la ventana usando la definición dada en (9.4).

$$I'(x, y) = \max \{I(x + i, y + j) \mid (i, j) \in R\} \quad (9.4)$$

R es la región definida por la ventana del filtro la cual es usualmente impar $p \times q$, con una secuencia ordenada de valores $p = (p_0, p_1, \dots, p_{K-1}, p_K, p_{K+1}, \dots, p_{2K})$, donde $K = \frac{pq+1}{2}$. Así, el valor máximo es p_{2K}



Figura 9.3: Funcionamiento principal del filtro mínimo con ventana de 3×3 . (a) La imagen I solo con ruido Sal y Pimienta al 0.01. (b) La imagen I' con filtrado mínimo . Nótese que se han enfatizado la Pimienta (Puntos negros).

Entre sus ventajas, es efectivo para eliminar el ruido negativo (puntos oscuros) y para preservar los bordes en las áreas brillantes de la imagen. Sin embargo, tiene la desventaja de aumentar el ruido positivo (puntos brillantes) y la generación de pérdidas de detalles en áreas oscuras. La Figura 9.4 muestra el efecto de usar el filtro máximo sobre una imagen con Sal y Primienta. Esto, como era de esperarse, enfatiza los puntos blancos de la pimienta.



Figura 9.4: Funcionamiento principal del filtro máximo con ventana de 3×3 . (a) La imagen I solo con ruido Sal y Pimienta al 0.01. (b) La imagen I' con filtrado máximo. Nótese que se han enfatizado la Sal (Puntos blancos).

Filtro de Rango

El filtro de rango calcula la resta entre el valor máximo y el valor mínimo de los píxeles dentro de una ventana alrededor de cada píxel, asignando esta diferencia como el nuevo valor del píxel como se muestra en (9.5). Este proceso resalta las variaciones de intensidad en el vecindario, lo que resulta útil para identificar bordes y transiciones bruscas en la imagen, lo que le hace adecuado en tareas de segmentación y detección de características, permitiendo distinguir con claridad diferentes regiones de la imagen.

$$I'(x, y) = \max \{I(x + i, y + j) \mid (i, j) \in R\} - \min \{I(x + i, y + j) \mid (i, j) \in R\} \quad (9.5)$$

donde R es la región definida por la ventana del filtro, usualmente de tamaño impar $p \times q$, y centrada en el píxel de interés (x, y) . A partir de los valores en R se selecciona el valor mínimo y valor máximo para luego ser restados como se indica en la expresión. En la Figura 9.5 se muestra un ejemplo de uso del filtro de rango.



Figura 9.5: Funcionamiento principal del filtro de rango con ventana de 3×3 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado de rango. Es detecta regiones donde hay cambios.

Filtro de Punto Medio

El filtro de punto medio calcula el promedio entre el valor mínimo y el valor máximo dentro de la ventana de filtrado. Para cada píxel de la imagen, el valor filtrado se obtiene mediante la definición mostrada en (9.6):

$$I'(x, y) = \frac{\min\{I(x+i, y+j) \mid (i, j) \in R\} + \max\{I(x+i, y+j) \mid (i, j) \in R\}}{2} \quad (9.6)$$

donde R es la región definida por la ventana del filtro, que generalmente es de tamaño $p \times q$ y centrada en el píxel de interés. La secuencia de valores en R se ordena de manera ascendente para encontrar el valor mínimo y de manera descendente para encontrar el valor máximo. Este filtro es útil para suavizar la imagen, ya que elimina el ruido sin perder demasiados detalles en áreas brillantes u oscuras. Sin embargo, puede introducir cierto desenfoque en las transiciones de intensidad. En la Figura 9.6 se muestra un ejemplo de uso del filtro de punto medio.



Figura 9.6: Funcionamiento principal del filtro de punto medio con ventana de 3×3 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado de punto medio .

9.6.3. Filtro Mediana

Es muy eficaz para eliminar el ruido “sal y pimienta” en la imagen. Su principio de funcionamiento consiste en reemplazar el nivel de gris de cada píxel por la mediana de los niveles de gris en una vecindad R del píxel de interés. Para lograrlo, se ordenan las intensidades cubiertas por la ventana, y se selecciona el valor central. Si la ventana cubre un número par de píxeles, se usa el promedio de dos valores. En general, se suelen usar ventanas impares para atenuar este inconveniente. Es imposible diseñar un filtro que elimine cualquier ruido pero que mantenga todas las estructuras importantes de la imagen intactas debido a que ninguno puede discriminar qué contenido de la imagen

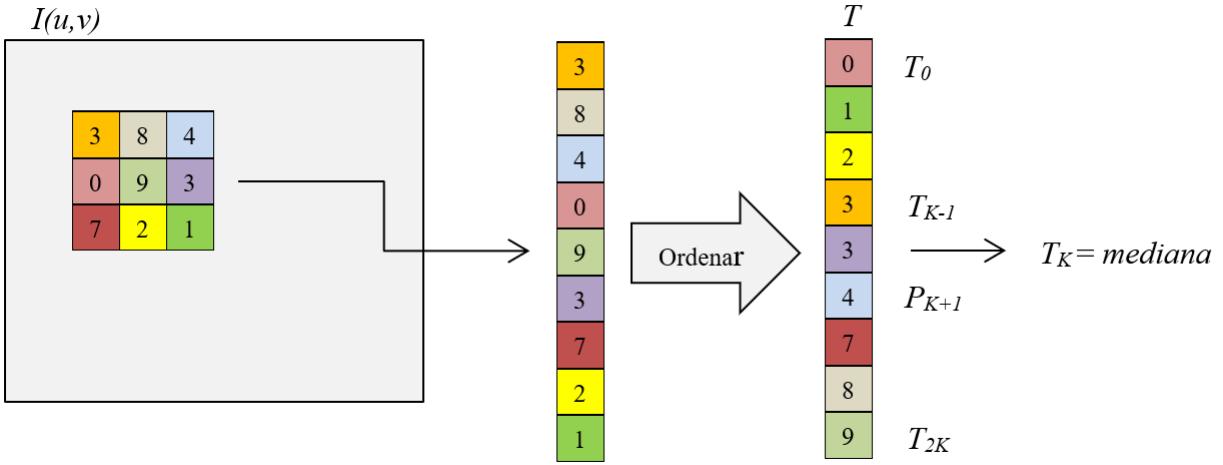


Figura 9.7: Ejemplo de cálculo de un filtro de mediana para una ventana de 3×3 píxeles.

es importante para el observador y cuál no lo es [2, 5, 13, 55]. El filtro mediana reemplaza cada píxel de la imagen por la mediana de los píxeles en la región del filtro R correspondiente. La definición formal se muestra en (9.7).

$$I'(x, y) = \text{median}\{I(x + i, y + j) \mid (i, j) \in R\}. \quad (9.7)$$

La mediana de una secuencia de $2K + 1$ valores de T_i es definida como el valor central T_K después de que $T = (T_0, \dots, T_{2K})$ está ordenada; es decir,

$$\text{median}(\underbrace{T_0, T_1, \dots, T_{K-1}}_{\text{valores de } K \leq T_K}, T_K, \underbrace{T_{K+1}, \dots, T_{2K}}_{\text{valores de } K \geq T_K}) \quad (9.8)$$

La Figura 9.7 ilustra el cálculo del filtro mediana con una vecindad de tamaño 3×3 píxeles con algunos valores de referencia.

En ((9.8)) se define la mediana de un conjunto de valores de tamaño impar, y si la longitud del lado de los filtros rectangulares es impar (que usualmente es el caso), entonces el número de elementos en la región de filtro es impar también. En este caso, el filtro mediana no crea ningunos valores de píxeles nuevos que no existían en la imagen original. Si, sin embargo, el número de elementos es par ($2K$ para algunos $K > 0$), entonces la mediana de la secuencia ordenada $T = (T_0, \dots, T_{2K-1})$ se define como la media aritmética de los dos valores medios T_{K-1} y T_K .

$$\text{median}(\underbrace{T_0, \dots, T_{K-1}}_{\text{valores de } K \leq T_K}, \underbrace{T_K, \dots, T_{2K-1}}_{\text{valores de } K \geq T_K}) = \frac{1}{2} \cdot (T_{K-1} + T_K). \quad (9.9)$$

El filtro de mediana es robusto frente al ruido impulsivo y puede preservar mejor los bordes y detalles finos en comparación con el filtro de media. Sin embargo, puede introducir cierto desenfoque en la imagen, especialmente en áreas donde hay una transición gradual de intensidad. La elección del tamaño de la ventana de filtrado es importante para equilibrar la eliminación del ruido y la preservación de los detalles. En la Figura 9.8 se muestra un ejemplo de uso del filtro de mediana, el cual es muy adecuado para eliminar Sal y Pimienta.



Figura 9.8: Funcionamiento principal del filtro mediana con ventana de 3×3 . (a) La imagen I solo con ruido Sal y Pimienta al 0.01. (b) La imagen I' con filtrado mediana. Nótese que bastante efectivo en la eliminación de los puntos Sal y Pimienta (blancos y negros).

9.6.4. Filtro Moda

Este es otro tipo de filtro no lineal utilizado para eliminar el ruido en imágenes. A diferencia del filtro de mediana, el filtro de moda reemplaza el valor del píxel central por el valor más frecuente (la moda) de los píxeles dentro de la ventana de filtrado. Matemáticamente, el filtro de moda se define como se aprecia en (9.10):

$$I'(u, v) = \text{moda}\{I(u + i, v + j) \mid (i, j) \in R\} \quad (9.10)$$

donde R denota la región del filtro (conjunto de coordenadas del filtro), generalmente una matriz de $p \times q$ píxeles.

El filtro de moda es particularmente eficaz para eliminar el ruido de "sal y pimienta.^{en}" las imágenes. A diferencia del filtro de mediana, que puede suavizar los bordes y detalles finos, el filtro de moda tiende a preservar mejor estos detalles, ya que selecciona el valor más frecuente en lugar de un valor promedio.

No obstante, el filtro de moda presenta ciertas limitaciones. En primer lugar, si el ruido predomina en la vecindad, podría seleccionar erróneamente un valor de ruido como la moda, lo que introduciría artefactos no deseados en la imagen. En segundo lugar, en caso de existir múltiples valores con la misma frecuencia máxima (es decir, múltiples modas), el filtro debe elegir arbitrariamente uno de ellos, lo cual puede generar sesgos en el resultado final, por lo anterior usualmente se toma el menor. En la Figura 9.10 se muestra un ejemplo de uso del filtro de moda, el cual es adecuado para segmentar regiones [2, 5, 13, 55].

9.6.5. Filtro Desviación Estándar

Resulta ser otro tipo de filtro no lineal que se utiliza para suavizar imágenes y reducir el ruido. A diferencia de los filtros de mediana y moda, el filtro de desviación estándar reemplaza el valor del píxel central por la desviación estándar de los valores de los píxeles dentro de la ventana de filtrado. Formalmente, el filtro de desviación estándar se define como se muestra en (9.11).

$$I'(u, v) = \sqrt{\frac{1}{|R|} \sum_{(i,j) \in R} (I(u + i, v + j) - \mu)^2} \quad (9.11)$$

donde R denota la región del filtro, $|R|$ es el número de píxeles en la región del filtro, e μ es la media de los valores de los píxeles dentro de la región del filtro, dada por:



Figura 9.9: Funcionamiento principal del filtro moda con ventana de 11×11 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado moda. Como se puede apreciar, esta tiene un aspecto artístico de pintura al óleo difuminado.

$$\mu = \frac{1}{|R|} \sum_{(i,j) \in R} I(u+i, v+j)$$

Este puede ser útil en ciertas aplicaciones, como la detección de bordes y la segmentación de imágenes. Al reemplazar cada píxel por la desviación estándar de su vecindad, este filtro resalta las regiones de la imagen donde hay una variación significativa en los valores de los píxeles, lo que puede indicar la presencia de bordes o límites entre diferentes objetos o regiones. Sin embargo, el filtro de desviación estándar también tiene algunas limitaciones. Puede ser sensible al ruido y a los valores atípicos, lo que puede introducir artefactos no deseados en la imagen filtrada. Además, al igual que otros filtros no lineales, puede distorsionar o eliminar detalles finos en la imagen [2, 5, 13, 55].

En el caso del filtro de desviación estándar, se trabaja con la muestra y no con la población, ya que generalmente se utiliza una región de tamaño reducido para estimar la dispersión de los valores de los píxeles.

La elección del filtro de desviación estándar depende de la aplicación específica y de las características de la imagen. En algunas situaciones, puede ser más adecuado utilizar otros filtros no lineales, como el filtro de mediana o el filtro de moda, para reducir el ruido sin introducir demasiada distorsión en la imagen. En la Figura 9.10 se muestra un ejemplo de uso del filtro de desviación estándar.



Figura 9.10: Funcionamiento principal del filtro desviación estándar con ventana de 3×3 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado desviación estándar. Resulta ser adecuado para detectar regiones que cambian, por lo que en este caso, enfatiza los bordes.

9.6.6. Filtro Entropía

El filtro de **entropía** mide la cantidad de desorden o aleatoriedad en los valores de los píxeles dentro de una ventana de filtrado. En (9.12) se muestra la manera de estimar la entropía para una región R .

$$I'(x, y) = \text{Entropía} \{I(x + i, y + j) \mid (i, j) \in R\} \quad (9.12)$$

donde R es la región definida por la ventana del filtro, usualmente de tamaño impar $p \times q$ y centrada en el píxel de interés (x, y) . La entropía del vecindario R se define como:

$$H(x, y) = - \sum_{i=1}^N p_i \log_2 p_i$$

donde N es el número de niveles de intensidad diferentes en el vecindario R , y p_i es la probabilidad de ocurrencia de cada intensidad I_i en R , considerando solo las que son diferentes de cero. Por lo anterior, para calcular la entropía, se recomienda construir el histograma de probabilidades de las intensidades dentro de la región R , lo que permite identificar las probabilidades cero y así evitar su inclusión en la suma total de la región.

La entropía, en este contexto, es una medida estadística que refleja la complejidad o variabilidad de los valores de intensidad en una región local, por lo que resulta útil para identificar texturas y patrones, segmentar imágenes y detectar algunos tipos de bordes. En la Figura 9.11 se muestra un ejemplo de uso del filtro entropía [2, 5, 13, 55].



Figura 9.11: Funcionamiento principal del filtro entropía con ventana de 3×3 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado entropía. Resulta ser adecuado identificar texturas en el reconocimiento de patrones.

9.6.7. Ejemplo en Python: Filtros espaciales

El código desarrollado permite visualizar de manera efectiva tanto filtros de orden (mínimo, máximo, mediana, moda y rango) como filtros estadísticos (desviación estándar y entropía), destacando particularmente estos últimos mediante técnicas de realce de contraste y mapas de color específicos. Esta metodología facilita la identificación de patrones estructurales, regiones de alta variabilidad y zonas con diferentes niveles de complejidad local, proporcionando herramientas valiosas para el análisis y caracterización de imágenes en aplicaciones como detección de bordes, segmentación y reconocimiento de patrones.

Solución

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from ip_functions import *
4
5 # Cargar imagen

```

```

6 I = plt.imread("bogota.jpg")
7 I = np.array(I)
8 S = 4 # Factor de escala
9 I = I[0::S, 0::S, :]
10
11 # Función para mejorar la visualización de filtros con valores pequeños
12 def enhance_contrast(img):
13     # Normalizar a 0-1
14     img_norm = (img - np.min(img)) / (np.max(img) - np.min(img) + 1e-10)
15     # Aplicar transformación gamma para mejorar el contraste
16     gamma = 0.5 # Valor menor a 1 para aumentar brillo en áreas oscuras
17     img_enhanced = np.power(img_norm, gamma)
18     # Escalar a 0-255
19     return (img_enhanced * 255).astype(np.uint8)
20
21 # Calcular los filtros de desviación estándar y entropía
22 std_img = stdfilt(I, (3,3), 'same', 'symmetric')
23 entropy_img = entropyfilt(I, (3,3), 'same', 'symmetric')
24
25 # Definir filtros en orden específico para mejorar la presentación visual
26 special_filters = {
27     "Orden Mínimo": ordfilt2(I, 1, np.ones((3,3)), 'same', 'symmetric'),
28     "Orden Máximo": ordfilt2(I, 9, np.ones((3,3)), 'same', 'symmetric'),
29     "Mediana": medfilt2(I, (3,3), 'same', 'symmetric'),
30     "Moda": modefilt(I, (5,5), 'same', 'symmetric'),
31     "Rango": rangefilt(I, (3,3), 'same', 'symmetric'),
32     "Desviación Estándar": enhance_contrast(std_img),
33     "Entropía": enhance_contrast(entropy_img)
34 }
35
36 # Crear figura con 4 filas y 2 columnas para mejor organización
37 fig, axes = plt.subplots(4, 2, figsize=(12, 16))
38 axes = axes.ravel()
39
40 # Mostrar la imagen original en la primera posición
41 axes[0].imshow(I)
42 axes[0].set_title("(a) Original")
43 axes[0].axis('off')
44
45 # Mostrar imágenes filtradas
46 labels = ["(b)", "(c)", "(d)", "(e)", "(f)", "(g)", "(h)"]
47 for i, (name, img) in enumerate(special_filters.items(), start=1):
48     # Usar 'plasma' para los filtros estadísticos - mejor visualización sin barra de color
49     if name in ["Desviación Estándar", "Entropía"]:
50         axes[i].imshow(img, cmap='plasma')
51     else:
52         axes[i].imshow(img)
53
54     axes[i].set_title(f"{labels[i-1]} {name}")
55     axes[i].axis('off')
56
57 plt.suptitle("Fig. 3: Filtros de orden y estadísticos aplicados a imagen urbana", fontsize=14)
58 plt.tight_layout()
59 plt.subplots_adjust(hspace=0.1, wspace=0.1)
60 plt.show()

```

En la Fig.9.12 se presentan ocho imágenes organizadas sistemáticamente para comparar los efectos de diversos filtros espaciales. La imagen (a) muestra la versión original sin alteraciones, conservando todos los detalles y texturas naturales. Las imágenes (b) y (c) exhiben los resultados de los filtros de orden mínimo y máximo, que destacan las regiones más oscuras y brillantes respectivamente dentro de cada vecindad. La imagen (d) con filtro de mediana elimina efectivamente el ruido mientras preserva los bordes, mientras que la (e) con filtro de moda resalta las intensidades más frecuentes. La imagen (f) muestra el rango de variación local de intensidades. Finalmente, las imágenes (g) y (h) visualizan mediante mapas de color las características estadísticas más complejas: la desviación estándar, que cuantifica la variabilidad local, y la entropía, que revela las áreas con mayor contenido de información y complejidad estructural. Las funciones referidas allí, se encuentran en el capítulo 13.

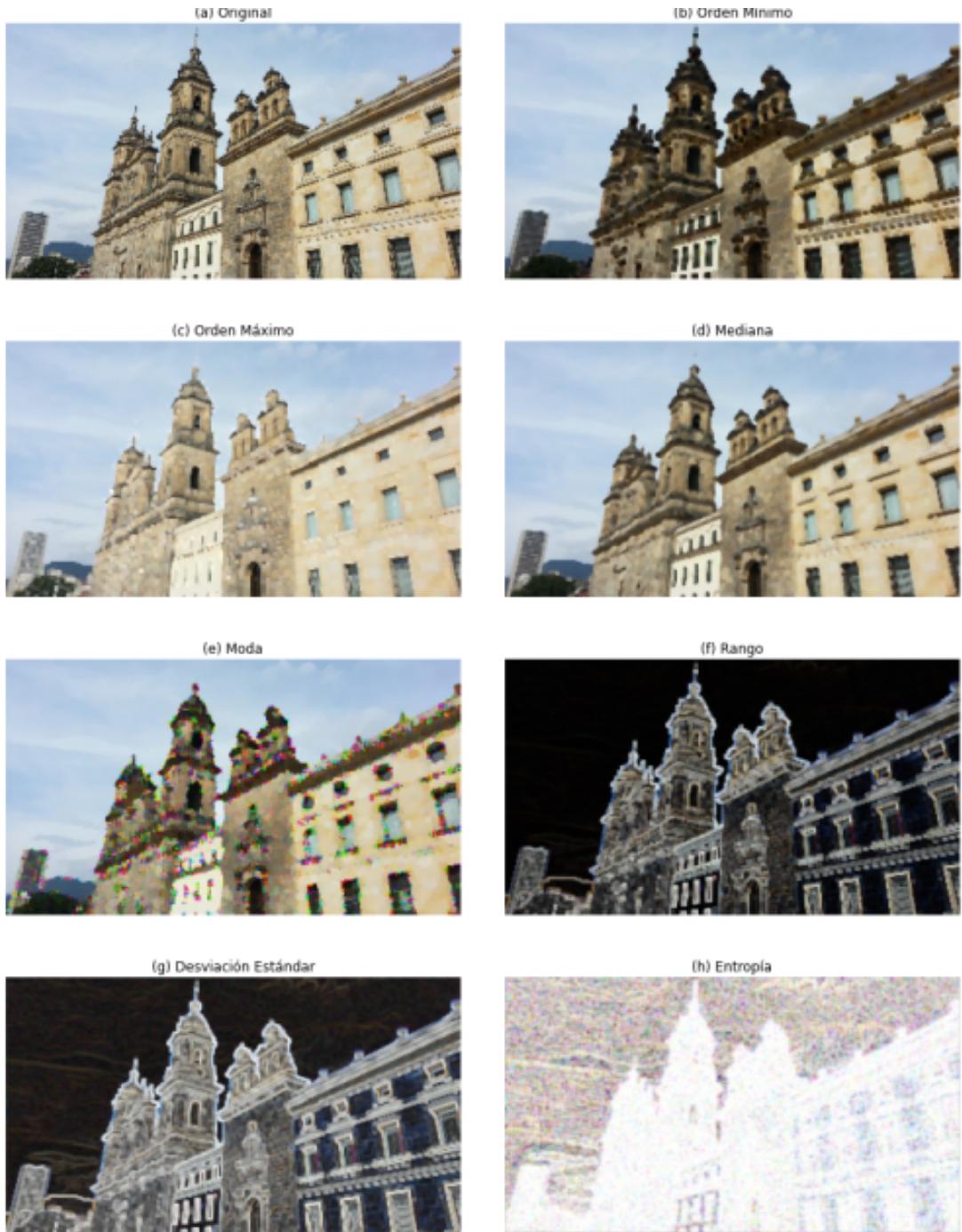


Figura 9.12: Filtros de procesamiento digital aplicados a la Catedral Primada de Bogotá. (a) Imagen original; (b) Orden mínimo que resalta zonas oscuras; (c) Orden máximo que enfatiza áreas brillantes; (d) Mediana que preserva bordes mientras reduce ruido; (e) Moda que destaca valores predominantes; (f) Rango que muestra diferencias locales de intensidad; (g) Desviación estandar que visualiza variabilidad local; (h) Entropía que identifica regiones con mayor complejidad informativa.

9.6.8. Filtro Gaussiano

El filtro Gaussiano lineal es ampliamente utilizado en el procesamiento de imágenes para suavizar y reducir el ruido. Su principio de funcionamiento se basa en aplicar un kernel Gaussiano K a la imagen, lo que produce un efecto de desenfoque suave y la eliminación de detalles de alta frecuencia (ruido). Matemáticamente, en el dominio espacial, el kernel Gaussiano bidimensional se expresa en 9.13:

$$K(x, y) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \quad (9.13)$$

donde σ es la desviación estándar de la función Gaussiana x y y son los límites del kernel respecto al centro, pudiendo ser estos positivos y negativos. Un valor más alto de σ produce un desenfoque más suave, mientras que un valor más bajo preserva mejor los bordes y los detalles de la imagen. La imagen filtrada $I'(x, y)$ se obtiene mediante la convolución de la imagen de entrada $I(x, y)$ con el kernel Gaussiano:

$$I'(x, y) = I(x, y) * K(x, y) \quad (9.14)$$

Este filtro se destaca por su suavizado isotrópico, lo que significa que el suavizado se realiza de manera uniforme en todas las direcciones, preservando mejor las características angulares de la imagen. Además, es eficiente en la eliminación de ruido Gaussiano y ruido de "sal y pimienta", y presenta un buen comportamiento en el dominio de la frecuencia, atenuando gradualmente las altas frecuencias sin introducir artefactos indeseados. Sin embargo, al igual que otros filtros, el filtro Gaussiano también tiene limitaciones. En la Figura 9.13 se muestra un ejemplo de uso del filtro de rango, el cual suaviza la imagen mejor que el promedio aritmético ya que no introduce artefactos [2,5,13,55].



Figura 9.13: Funcionamiento principal del filtro Gaussiano con ventana de 3×3 y desviación de 0.5. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Gaussiano. Esta suaviza la imagen y se ajusta más al modelo de desenfoque por plano de formación en imágenes que el promedio.

Si se elige un valor de σ demasiado grande, puede provocar una pérdida excesiva de detalles finos y un desenfoque excesivo de la imagen. Por lo tanto, la elección adecuada del valor de σ es crucial para lograr un equilibrio óptimo entre la reducción de ruido y la preservación de detalles en la imagen filtrada. Por lo anterior, se recomienda hacer el tamaño del kernel Gaussiano con un valor como mínimo 6 veces el valor de la desviación estándar a usar. La Figura 9.14 muestra el aspecto de un kernel Gaussiano 3D de 31×31 y con desviación estándar de $\sigma = 5$, el cual se parecía es simétrico.

Cuando se usa el kernel Gaussiano, es necesario normalizar el resultado a 1 por ser un filtro pasa bajo:

$$k_s = \sum_{x=1}^n \sum_{y=1}^m K(x, y)$$

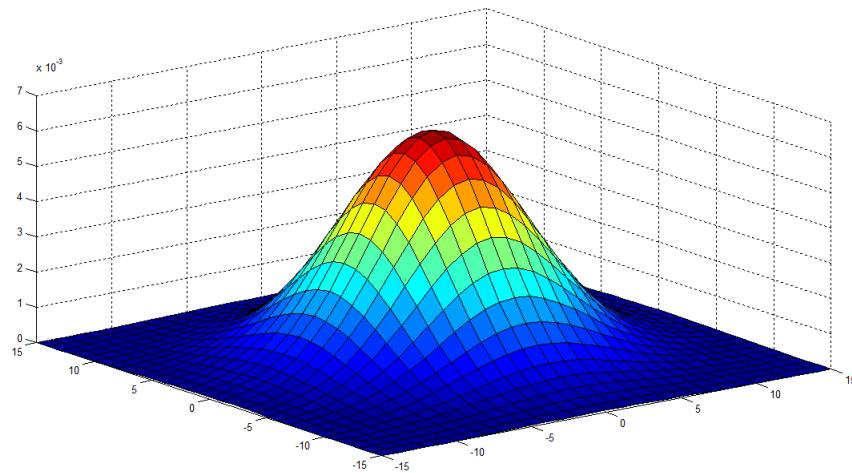


Figura 9.14: Visualización 3D del kernel Gaussiano de 31×31 y con desviación estándar de $\sigma = 5$.

Se tiene entonces que el Gaussiano normalizado a 1 será el siguiente:

$$G_u(x, y) = \frac{1}{2\pi\sigma^2 k_s} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (9.15)$$

Ejemplo de Cálculo de las posiciones del kernel Gaussiano

Para calcular el kernel Gaussiano 5×5 con desviación estándar $\sigma = 0.5$ y centrado en $(0, 0)$, se usa la expresión reducida (9.16), teniendo en cuenta que es necesaria su normalización a 1:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (9.16)$$

donde:

- $G(x, y)$ es el valor del kernel en la posición (x, y)
- σ es la desviación estándar (en este caso, 0.5)
- π es la constante pi (aproximadamente 3.14159)

Se calculan los valores para cada posición del kernel 5×5 como se indica en la Tabla 9.1. Nótese que se omitieron los valores negativos por la simetría en la respuesta al estar estos al cuadrado en la ecuación donde se evalúa:

Posición (x,y)	Cálculo	$G(x, y)$	$G_u(x, y)$
(0,0)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{0^2+0^2}{2(0.5)^2}\right)$	0.6366	0.6187
(0,1)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{0^2+1^2}{2(0.5)^2}\right)$	0.0862	0.0837
(0,2)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{0^2+2^2}{2(0.5)^2}\right)$	0.0023	0.0002
(1,0)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{1^2+0^2}{2(0.5)^2}\right)$	0.0862	0.0837
(1,1)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{1^2+1^2}{2(0.5)^2}\right)$	0.0117	0.0113
(1,2)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{1^2+2^2}{2(0.5)^2}\right)$	0.0003	0.0002
(2,0)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{2^2+0^2}{2(0.5)^2}\right)$	0.0023	0.0002
(2,1)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{2^2+1^2}{2(0.5)^2}\right)$	0.0003	0.0002
(2,2)	$\frac{1}{2\pi(0.5)^2} \exp\left(-\frac{2^2+2^2}{2(0.5)^2}\right)$	1.7911×10^{-7}	0.0000

Cuadro 9.1: Cálculos para las posiciones únicas en el cuadrante inferior derecho de la matriz 5×5 con desviación estándar de 0.5 para el Gaussiano.

Luego, se normalizan los valores para que la suma sea igual a 1 dividiendo por la suma de la matriz inicial. Por lo tanto, el kernel Gaussiano 5×5 con desviación estándar 0.5 y centrado en (0, 0) es:

$$G(x, y) = \begin{bmatrix} 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0002 & 0.0837 & 0.6187 & 0.0837 & 0.0002 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \end{bmatrix} \quad (9.17)$$

Si bien se calculó sobre 5×5 , era suficiente en una ventana de 3×3 porque esta converge rápidamente a cero.

9.6.9. Ejemplo en Python: Filtros pasa bajo

Este código ilustra la aplicación de filtros pasa bajos en el procesamiento de imágenes, utilizando técnicas de suavizado como los filtros de promedio y gaussiano. Estos atenúan el ruido y realzan las estructuras de baja frecuencia, permitiendo una mejor percepción de las variaciones suaves en la imagen.

Solución

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from ip_functions import * # Asegúrate de que esta biblioteca esté disponible
4
5 # Cargar imagen
6 I = plt.imread("bogota.jpg")
7 I = np.array(I)
8 S = 4 # Factor de escala
9 I = I[0::S, 0::S, :]
10
11 # Definir filtros pasa bajos
12 low_pass_filters = {
13     "Promedio": fspecial('average'),

```



Figura 9.15: Se presentan tres imágenes: (a) la imagen original, (b) el resultado del filtrado con un filtro de promedio y (c) el resultado del filtrado con un filtro gaussiano.

```

14     "Gaussiano": fspecial('gaussian')
15 }
16
17 # Aplicar filtros pasa bajos
18 filtered_images = {name: imfilter(I, k, 'same', 'symmetric') for name, k in low_pass_filters.items()}
19
20 # Crear figura y mostrar imágenes en un solo subplot
21 num_filters = len(filtered_images)
22 cols = 3 # Tres columnas
23 rows = (num_filters // cols) + (num_filters % cols > 0) # Filas necesarias
24 fig, axes = plt.subplots(rows, cols, figsize=(12, 4))
25 axes = axes.ravel()
26
27 # Mostrar la imagen original en la primera posición
28 axes[0].imshow(I, cmap='gray')
29 axes[0].set_title("Original")
30 axes[0].axis('off')
31
32 # Mostrar imágenes filtradas
33 for i, (name, img) in enumerate(filtered_images.items(), start=1):
34     axes[i].imshow(img, cmap='gray')
35     axes[i].set_title(name)
36     axes[i].axis('off')
37
38 # Ocultar subplots vacíos si hay
39 for j in range(i + 1, rows * cols):
40     axes[j].axis('off')
41
42 plt.suptitle("Filtros Pasa Bajos", fontsize=14)
43 plt.tight_layout()
44 plt.show()

```

En la Fig.9.15 se presentan tres imágenes: la imagen original, un filtrado con un filtro de promedio y un filtrado con un filtro gaussiano. La imagen original muestra todos los detalles sin alteraciones. La segunda imagen, con el filtro promedio, suaviza la imagen eliminando variaciones rápidas de intensidad. La tercera imagen, con el filtro gaussiano, logra un efecto similar al filtro promedio pero con una transición más suave, preservando mejor los contornos. Las funciones referidas allí, se encuentran en el capítulo 13.

9.7. Filtros de Detector de Bordes (Pasa-Alto)

Las derivadas en imágenes se relacionan con la tasa de cambio de los valores de píxeles en diferentes direcciones. En una imagen digital, que es una función discreta bidimensional, las derivadas se aproximan mediante diferencias finitas entre píxeles adyacentes como se muestra en la Tabla 9.2. Estas diferencias capturan los cambios abruptos en intensidad, lo que resulta útil para detectar bordes y texturas. Este concepto es fundamental en el procesamiento de imágenes y la visión por computadora, aplicándose en técnicas como la detección de bordes, la segmentación y el realce de imágenes. Por otro lado, las derivadas de orden superior pueden proporcionar información complementaria sobre la estructura local de la imagen, como la curvatura de los bordes o las transiciones.

Tipo de Derivada	Dirección	Representación
Frontal	x	$\frac{\partial I(x,y)}{\partial x} \approx I(x+1,y) - I(x,y)$ Kernel: $[0 \quad -1 \quad 1]$
	y	$\frac{\partial I(x,y)}{\partial y} \approx I(x,y+1) - I(x,y)$ Kernel: $\begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$
Trasera	x	$\frac{\partial I(x,y)}{\partial x} \approx I(x,y) - I(x-1,y)$ Kernel: $[-1 \quad 1 \quad 0]$
	y	$\frac{\partial I(x,y)}{\partial y} \approx I(x,y) - I(x,y-1)$ Kernel: $\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$
Central	x	$\frac{\partial I(x,y)}{\partial x} \approx \frac{I(x+1,y) - I(x-1,y)}{2}$ Kernel: $[-1 \quad 0 \quad 1]$
	y	$\frac{\partial I(x,y)}{\partial y} \approx \frac{I(x,y+1) - I(x,y-1)}{2}$ Kernel: $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$

Cuadro 9.2: Definición discreta de las derivadas frontal, trasera y central.

Un borde se puede definir como un conjunto de píxeles conectados que forman un límite entre dos regiones separadas. El método de detección de bordes transforma la imagen original en una imagen de solo fronteras con la ayuda de algunos operadores para así reducir la cantidad de información y solo conservar las propiedades estructurales de una imagen. Para lograr la detección de bordes, la imagen se convierte a escala de grises y, posteriormente, se le aplica el operador para extraer los límites presentes en una imagen de salida.

En una imagen, la primera derivada se aproxima mediante la diferencia entre píxeles adyacentes. Las transiciones de blanco a negro y de negro a blanco son ejemplos claros de cómo funciona la primera derivada en imágenes como se muestra en la Figura 9.16. En una transición de blanco a negro, la primera derivada es negativa, la cual alcanza su valor máximo negativo en el borde más pronunciado. Por el contrario, en una transición de negro a blanco, la primera derivada es positiva. En ambos casos, los valores más prominentes de la derivada indican cambios abruptos en la intensidad.

La detección de bordes resulta ser una herramienta esencial de segmentación a partir de la detección de discontinuidades significativas en la imagen las cuales podrían representar objetos en ella, o ser útil a la hora de segmentar las imágenes para poder observar las distintas regiones que hay en ella. Existen 3 tipos de bordes:

- Bordes horizontales.
- Bordes verticales.
- Bordes diagonales.

Existen diversas técnicas disponibles para detectar los bordes tales como los operadores Roberts, Prewitt, Sobel, Laplacian de Gaussian y Canny. Estas se describen a continuación.

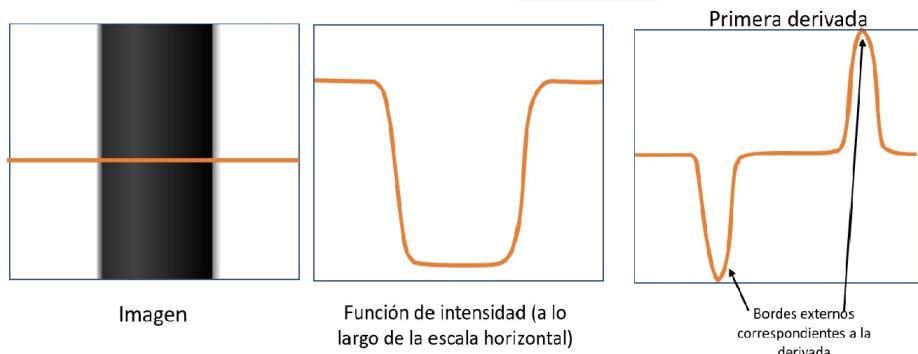


Figura 9.16: Imagen primera derivada de una transición de blanco a negro y de negro a blanco.

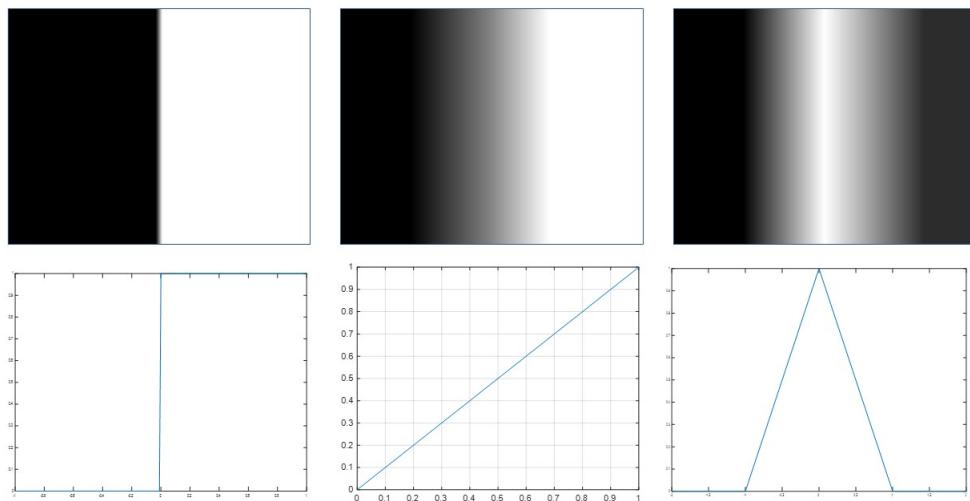


Figura 9.17: Ejemplos de bordes y su respectiva función. Se muestra una imagen con un cambio brusco de intensidad, otra con un degradado gradual y otra con dos zonas de degradado.

9.7.1. El Gradiente

Este concepto se refiere a la magnitud y dirección de los cambios en la intensidad de los píxeles de una imagen. Su cálculo se realiza a tomando la derivada de la imagen en cada punto en dos direcciones perpendiculares, generalmente en forma horizontal y vertical. El gradiente resulta útil para detectar bordes y contornos, ya que los límites a menudo corresponden a cambios bruscos en la intensidad de los píxeles. El gradiente se puede calcular utilizando varios operadores, como el Sobel, el Prewitt y el de Laplaciano, entre otros [2,5,13,55].

En una dimensión, un borde escalonado está asociado con un pico local en la primera derivada. El gradiente es una medida de cambio en una función, y una imagen puede considerarse como una matriz de muestras de alguna función continua de la imagen. La Figura 9.17 presenta ejemplos de transiciones de bordes en imágenes: una con un cambio brusco, otra con un degradado lineal y, por último, una con dos zonas de degradado ascendente y descendente.

Por analogía, los cambios significativos en los valores de gris de una imagen se pueden detectar utilizando una aproximación discreta. El gradiente es el equivalente bidimensional de la primera derivada y se define como el vector mostrado en (9.18).

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad (9.18)$$

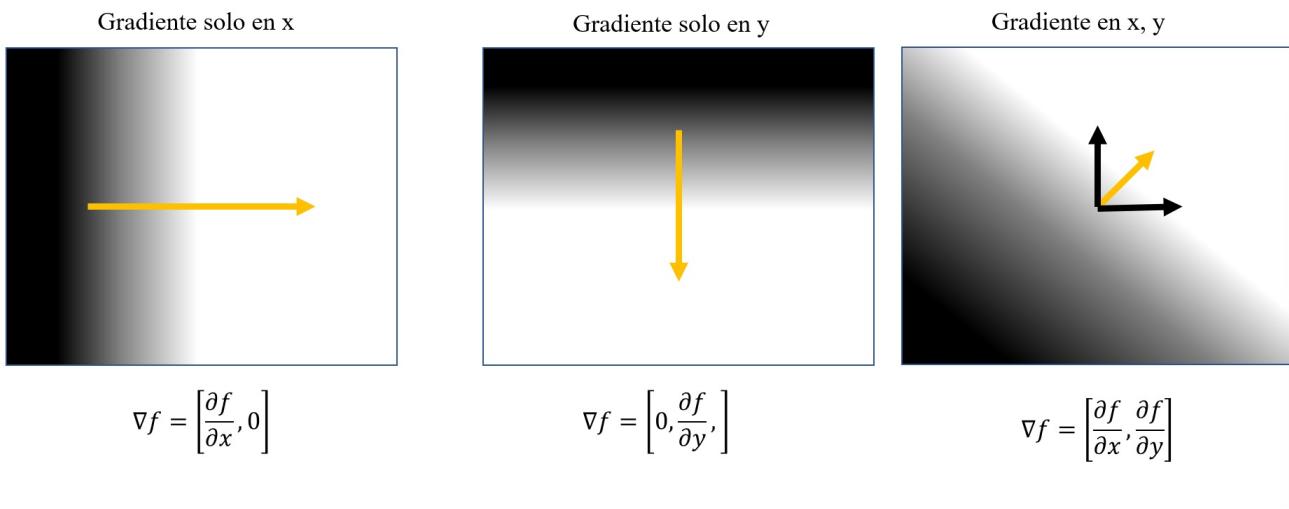


Figura 9.18: Dirección del gradiente, magnitud y sentido. Se aprecian uno horizontal, vertical y diagonal.

La magnitud del gradiente se calcula sumando las derivadas en las direcciones horizontal y vertical. Una vez que se han calculado los gradientes en cada punto de la imagen, se pueden utilizar técnicas para detectar bordes y contornos, como se indica en (9.19):

$$|G[f(x, y)]| = \sqrt{G_x^2 + G_y^2} \quad (9.19)$$

Sin embargo, es una práctica común aproximar la magnitud del gradiente por valores absolutos como se muestra en (9.20) y (9.21):

$$G[f(x, y)] \simeq |G_x| + |G_y| \quad (9.20)$$

$$G[f(x, y)] \simeq \max(|G_x|, |G_y|) \quad (9.21)$$

A partir del análisis vectorial, la dirección del gradiente se define como se aprecia en (9.22):

$$\alpha(x, y) = \tan^{-1} \left(\frac{G_y}{G_x} \right) \quad (9.22)$$

La Figura 9.18 ilustra posibles gradientes con algunas direcciones.

Se basa en la aproximación de la derivada mediante una diferencia finita. Para calcular el gradiente en una dirección determinada, se utilizan (9.23) y (9.24), donde $I(x, y)$ es el valor de la intensidad píxel en la posición (x, y) de la imagen.

$$G_x(x, y) = I(x+1, y) - I(x-1, y) \quad (9.23)$$

$$G_y(x, y) = I(x, y+1) - I(x, y-1) \quad (9.24)$$

9.7.2. El Laplaciano

La principal diferencia entre Laplaciano y otros operadores como Prewitt, Sobel, y Robert, que se mencionan más adelante, es que todos estos son kernels derivativas de primer orden, pero, por el contrario, el Laplaciano es de segundo orden. Otra diferencia entre Laplaciano y otros operadores es que no resalta los bordes en ninguna dirección en particular, sino que los destaca hacia afuera o adentro, dependiendo del operador utilizado [2, 5, 13, 55]. Se deben destacar que el filtro Laplaciano es sensible al ruido, por lo que no funciona adecuadamente si hay ruido dominante en la imagen. Debido a este hecho, se sugiere primero aplicar un desenfoque Gaussiano para suavizar la imagen y así lograr que el Laplaciano sea más efectivo al atenuar algunos tipos de ruido.

Algunas propiedades del Laplaciano son:

- A diferencia de los filtros de primer orden que requieren dos kernel para encontrar los bordes, el Laplaciano usa solo uno.
- Es isotrópico, es decir, produce una magnitud del borde uniformemente en todas las direcciones.
- En contraste con los filtros de primer orden, pierde la información correspondiente a la orientación del borde.
- Proporciona una mejor localización de bordes en comparación con los de primer orden.

Laplaciano de 4 Puntos

El Laplaciano de cuatro puntos es un operador utilizado para detectar cambios bruscos o bordes en una imagen. Este opera a partir de la segunda derivada de la imagen y se utiliza para resaltar las regiones donde hay cambios significativos en los niveles de intensidad de los píxeles adyacentes. Este es una buena opción cuando se necesita la detección de bordes simple, eficiente y con poco ruido, especialmente en imágenes con patrones rectangulares o cuadrados.

Obtención del Kernel Laplaciano de 4 Puntos

La aproximación hacia adelante para la primera derivada en una dimensión se define como:

$$\frac{\partial I}{\partial x} \approx \frac{I(x+h) - I(x)}{h}$$

donde h es el tamaño del paso en la dirección de x .

La aproximación hacia atrás para la primera derivada en una dimensión se define como:

$$\frac{\partial I}{\partial x} \approx \frac{I(x) - I(x-h)}{h}$$

Para obtener la aproximación central para la segunda derivada, se restan la aproximación hacia adelante de la aproximación hacia atrás:

$$\frac{\partial^2 I}{\partial x^2} \approx \frac{I(x+h) - I(x)}{h} - \frac{I(x) - I(x-h)}{h}$$

Simplificando, se obtiene para x :

$$\frac{\partial^2 I}{\partial x^2} \approx \frac{I(x+h) - 2I(x) + I(x-h)}{h^2}$$

De manera análoga se obtiene para y :

$$\frac{\partial^2 I}{\partial y^2} \approx \frac{I(x, y + h) - 2I(x, y) + I(x, y - h)}{h^2}$$

Para llegar al operador Laplaciano de 4 puntos a partir de las segundas derivadas parciales discretizadas, se suman las segundas derivadas respecto a x y y :

$$\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \approx \frac{I(x + h, y) + I(x, y + h)}{h^2} - 2 \frac{I(x, y)}{h^2} + \frac{I(x - h, y) + I(x, y - h)}{h^2}$$

La expresión es una aproximación del Laplaciano de la función I en el punto (x, y) utilizando el operador de cuatro puntos:

$$\nabla^2 I(x, y)_4 \approx \frac{I(x + h, y) + I(x - h, y) + I(x, y + h) + I(x, y - h) - 4I(x, y)}{h^2}$$

La elección de $h = 1$ en la expresión discreta del Laplaciano de 4 puntos es una convención común en el procesamiento de imágenes digitales, puesto que se supone que este es el tamaño de paso entre píxeles adyacentes, lo cual simplifica los cálculos y la implementación del algoritmo:

$$\nabla^2 I(x, y)_4 \approx I(x + 1, y) + I(x - 1, y) + I(x, y + 1) + I(x, y - 1) - 4I(x, y)$$

Sustituyendo las aproximaciones discretas de las segundas derivadas parciales en la definición del operador Laplaciano, se llega a la expresión discreta del kernel Laplaciano de 4 puntos ilustrada en (9.25):

$$\nabla^2 I(x, y)_4 \approx \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (9.25)$$

Nótese que la sumatoria de todos los términos del kernel Laplaciano de cuatro puntos, por ser un filtro pasa alto, es igual a cero.

El factor de ajuste alfa (α) se introduce en el kernel del Laplaciano de 4 puntos para modificar su sensibilidad en la detección de bordes. El kernel modificado resulta como se aprecia en (9.26):

$$\nabla^2 I(x, y)_4 \approx \frac{1}{4(\alpha + 1)} \begin{bmatrix} \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \\ \frac{1-\alpha}{4} & -1 & \frac{1-\alpha}{4} \\ \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \end{bmatrix} \quad (9.26)$$

donde α es un parámetro que controla la sensibilidad del detector de bordes siendo su objetivo:

- **Ajustar la sensibilidad:** Un valor grande de α hace que el detector sea más sensible a los cambios de intensidad, lo que puede ser útil para detectar bordes débiles o en imágenes con ruido.
- **Reducir el ruido:** Un valor menor de α puede reducir el ruido en la imagen, ya que disminuye la sensibilidad del detector a cambios menores en la intensidad.
- **Mejorar la robustez:** α puede mejorar la robustez del detector de bordes ante cambios en la iluminación o en la textura de la imagen.

En MATLAB, el valor predeterminado de α es 0.2 puesto que funciona bien en una amplia variedad de imágenes y condiciones sin introducir muchos artefactos de ruido. Sin embargo, el usuario puede ajustar α según sea necesario para su aplicación específica. En la Figura 9.19 se muestra un ejemplo de uso del filtro de Laplaciano de cuatro puntos.



Figura 9.19: Funcionamiento principal del filtro Laplaciano de 4 puntos con ventana de 3×3 y $\alpha = 0$. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Laplaciano de 4 . Es adecuada para encontrar border en las imágenes, aun sueve usarse en ocmpaía de otros filtros como el Gaussiano ya que enfatiza el ruido.

Laplaciano de 8 Puntos

Este kernel es más sensible a los cambios en las diagonales que el kernel de Laplaciano estándar de 4 puntos, lo que lo hace útil para detectar bordes y contornos en imágenes con patrones diagonales o curvos.

Obtención del Kernel Laplaciano de 8 Puntos

La expresión original del Laplaciano de 4 puntos es:

$$\nabla^2 f(x, y)_4 = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2}$$

Se sabe, del Laplaciano de 4 puntos anterior, que:

$$\nabla^2 I(x, y)_4 \approx I(x+1, y) + I(x-1, y) + I(x, y+1) + I(x, y-1) - 4I(x, y)$$

Para el Laplaciano de 8 puntos, se suman las derivadas parciales mixtas a la expresión original :

$$\nabla^2 f(x, y)_8 = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2} + \frac{\partial^2 I(x, y)}{\partial x \partial y}_{y=x} + \frac{\partial^2 f Ix, y}{\partial y \partial x}_{y=-x}$$

Entonces:

$$\nabla^2 f(x, y)_8 = \nabla^2 I(x, y)_4 + \frac{\partial^2 I(x, y)}{\partial x \partial y}_{y=x} + \frac{\partial^2 f Ix, y}{\partial y \partial x}_{y=-x}$$

La derivada mixta de segundo orden (aproximada) para la diagonal $y = -x$:

$$\frac{\partial^2 I}{\partial x \partial y}_{y=x} \approx I(x+1, y-1) - 2I(x, y) + I(x-1, y+1)$$

Y para la diagonal $y = x$, corresponde a:

$$\frac{\partial^2 I}{\partial x \partial y} \Big|_{y=-x} \approx I(x+1, y+1) - 2I(x, y) + I(x-1, y-1)$$

Sumando los términos:

$$\begin{aligned} \nabla^2 f(x, y)_8 &\approx [I(x+1, y) + I(x-1, y) + I(x, y+1) + I(x, y-1) - 4I(x, y)] + \dots \\ &+ [I(x+1, y-1) - 2I(x, y) + I(x-1, y+1)] + [I(x+1, y+1) - 2I(x, y) + I(x-1, y-1)] \end{aligned}$$

Simplificando:

$$\begin{aligned} \nabla^2 f(x, y)_8 &\approx I(x+1, y) + I(x-1, y) + I(x, y+1) + I(x, y-1) + I(x+1, y-1) + \dots \\ &+ I(x-1, y+1) + I(x+1, y+1) + I(x-1, y-1) - 8I(x, y) \end{aligned}$$

Dado que las contribuciones diagonales suelen simplificarse en términos prácticos, la expresión resultante puede combinarse y simplificarse aún más en un kernel de convolución práctico para procesamiento de imágenes de ocho puntos como se muestra en (9.27).

$$\nabla^2 f(x, y)_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (9.27)$$

Nótese que la sumatoria de todos los términos del kernel Laplaciano de ocho puntos, por ser un filtro pasa alto, es igual a cero.

Este kernel representa una combinación de las derivadas parciales en todas las direcciones, incluidas las diagonales. En la Figura 9.20 se muestra un ejemplo de uso del filtro de Laplaciano de ocho puntos.



Figura 9.20: Funcionamiento principal del filtro Lapaciano de 4 puntos con ventana de 3×3 y $\alpha = 0$. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Lapaciano de 4. Es adecuada para encontrar bordes en las imágenes, al usarse acompañado de otros filtros como el Gaussiano, ya que por sí solo enfatiza el ruido.

Este filtro permite una detección de bordes más completa y precisa en todas las direcciones; esto resulta en una mayor sensibilidad a los cambios de intensidad en cualquier orientación, mejora la capacidad para resaltar detalles finos y texturas complejas, reduce los artefactos direccionales y proporciona una respuesta isotrópica que representa fielmente las características de la imagen.

9.7.3. Filtro Laplaciano del Gaussiano

Debido a que el filtro Laplaciano viene generalmente asociado con el Gaussiano, se suele definir un filtro nuevo denominado Laplaciano sobre el filtro Gaussiano (LoG – Laplacian of Gaussian) para realizar ambas labores en una sola pasada [5, 13]. Este se utiliza para detectar bordes y características en una imagen. Este operador se obtiene aplicando el operador Laplaciano a un filtro Gaussiano. El proceso se divide en dos etapas para lograr este resultado. En la primera etapa, se aplica un filtro Gaussiano a la imagen el cual suaviza la imagen al eliminar detalles finos y el ruido no deseado, lo que permite tener una versión más homogénea de la imagen, eliminando texturas y variaciones pequeñas en una superficie. En la segunda etapa, se estima el Laplaciano sobre la imagen suavizada, el cual resalta los cambios bruscos de intensidad, identificando así los bordes. Al combinar el suavizado del filtro Gaussiano y el cálculo del Laplaciano, se obtiene una representación que resalta las áreas donde ocurren cambios rápidos de intensidad sin los efectos indeseados del ruido emergente. Sin embargo, es importante tener en cuenta que el LoG puede ser computacionalmente costoso y depende de parámetros como el tamaño de la ventana, la desviación estándar y el umbral de detección. Además, es posible que genere una detección excesiva de bordes si se superponen dos bordes en diferentes escalas.

A partir de la definición del Laplaciano mostrado en (9.28):

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \quad (9.28)$$

Y del Gaussiano definido en (9.29) :

$$k(x, y) = k(x)k(y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (9.29)$$

Se calcula la primera derivada parcial del Gaussiano en (9.29) con respecto a x :

$$\frac{\partial k(x, y)}{\partial x} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(\frac{1}{2\sigma^2} \right) (-2x)$$

$$\frac{\partial k(x, y)}{\partial x} = -\frac{y}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Luego, su segunda derivada parcial con respecto a x :

$$\frac{\partial^2 k(x, y)}{\partial x^2} = \frac{\partial}{\partial x} \left(-\frac{x}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \right)$$

$$\frac{\partial^2 k(x, y)}{\partial x^2} = -\frac{1}{2\pi\sigma^4} \left(e^{-\frac{x^2+y^2}{2\sigma^2}} + \frac{x}{2\sigma^2} (-2x) e^{-\frac{x^2+y^2}{2\sigma^2}} \right)$$

Se llega a la segunda deriva con respecto a x en (9.30):

$$\frac{\partial^2 k(x, y)}{\partial x^2} = -\frac{1}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(1 - \frac{x^2}{\sigma^2} \right) \quad (9.30)$$

Debido a la simetría en (9.29) con respecto a x , la dependencia de la ecuación respecto a la variable y es similar. Por lo anterior, la segunda derivada con respecto a y resulta como (9.31):

$$\frac{\partial^2 k(x, y)}{\partial y^2} = -\frac{1}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(1 - \frac{y^2}{\sigma^2}\right) \quad (9.31)$$

Seguidamente, reemplazando los resultados de las segundas derivadas obtenidas en (9.30) y (9.31) en (9.28):

$$\begin{aligned} \nabla^2 k(x, y) &= \frac{\partial^2 k(x, y)}{\partial x^2} + \frac{\partial^2 k(x, y)}{\partial y^2} \\ \nabla^2 k(x, y) &= -\frac{1}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(1 - \frac{x^2}{\sigma^2}\right) - \frac{1}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \left(1 - \frac{y^2}{\sigma^2}\right) \\ \nabla^2 k(x, y) &= e^{-\frac{x^2+y^2}{2\sigma^2}} \left(-\frac{1}{2\pi\sigma^4} + \frac{x^2}{2\pi\sigma^6} - \frac{1}{2\pi\sigma^4} + \frac{y^2}{2\pi\sigma^6}\right) \\ \nabla^2 k(x, y) &= e^{-\frac{x^2+y^2}{2\sigma^2}} \left(-\frac{2}{2\pi\sigma^4} + \frac{x^2+y^2}{2\pi\sigma^6}\right) \\ \nabla^2 k(x, y) &= e^{-\frac{x^2+y^2}{2\sigma^2}} \left(-\frac{1}{\pi\sigma^4} + \frac{x^2+y^2}{2\pi\sigma^6}\right) \end{aligned}$$

Ordenando:

$$\nabla^2 k(x, y) = -\frac{1}{\pi\sigma^4} \left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Finalmente, la versión discreta del LoG corresponde a la mostrada en (9.32):

$$\nabla^2 k(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] \quad (9.32)$$

En la Figura 9.21 se muestra el kernel Laplaciano del Gaussiano, el cual es un sombrero invertido:

Nótese que $G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ es la Gaussiana:

$$\nabla^2 k(x, y) = G(x, y) \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] \quad (9.33)$$

La ecuación (9.32) es la respuesta clásica para el Laplaciano del Gaussiano. Sin embargo, esta no es útil para aplicaciones prácticas, puesto que es necesario normalizar la matriz de (9.33) a 0 como sucede con los filtros pasa alto como el Lapaciano de cuatro y ocho puntos. Esto se logra restando el promedio de la matriz, denotado como k_l , a cada uno de los valores, como se muestra en (9.35) usando (9.34):

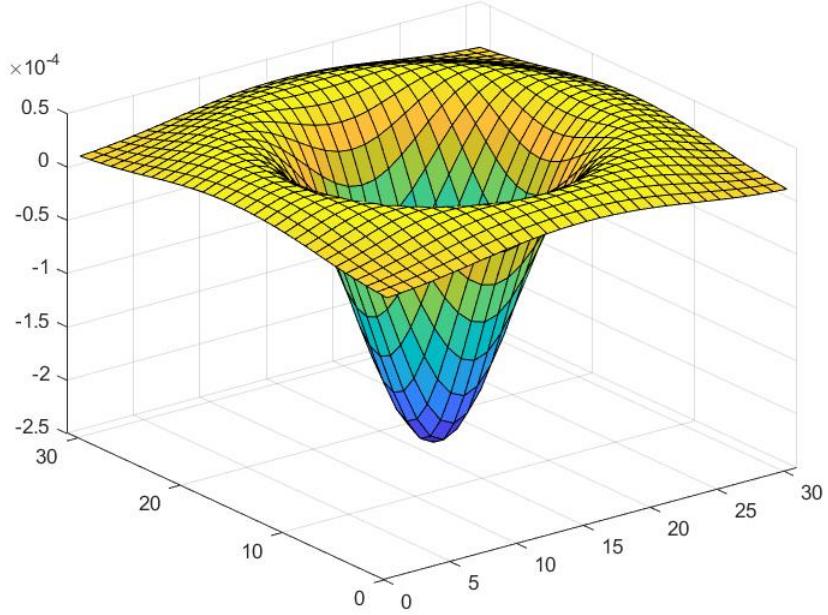


Figura 9.21: Kernel LoG de 31x31 y con $\sigma = 5$.

$$k_l = \frac{1}{nm} \sum_{x=1}^n \sum_{y=1}^m \nabla^2 k(x, y) \quad (9.34)$$

donde n y m son las dimensiones de la matriz normalizada se obtiene el LoG Normalizado a cero por ser un pasa alto:

$$\nabla_n^2 k(x, y) = \nabla^2 k(x, y) - k_l \quad (9.35)$$

El Laplaciano de Gaussiana (LoG) ofrece ventajas sobre el uso del Laplaciano porque usa el suavizado previo con la detección de bordes, lo que mejora la precisión en la identificación de bordes. Al aplicar primero una Gaussiana para suavizar la imagen, se reducen las variaciones y el ruido de alta frecuencia que el operador Laplaciano por sí solo amplificaría debido a su naturaleza de segundo derivada, lo que permite que el LoG detectar bordes reales al eliminar los bordes falsos causados por el ruido, resultando en una detección mejorada. En la Figura 9.22 se muestra un ejemplo de uso del filtro de Laplaciano del Gaussiano.

Ejemplo de Cálculo de las posiciones del kernel Laplaciano de Gaussiana (LoG)

Calcular el Laplaciano de Gaussiana (LoG) para un kernel de 5×5 , usando una desviación estándar $\sigma = 0.5$.

Solución

Se definen y calculan los parámetros básicos usar:

$$\sigma = 0.5 \quad \sigma^2 = 0.25 \quad \sigma^4 = 0.0625$$

Por otro lado, solo se determinan las posiciones únicas, ya que el kernel es simétrico y los valores negativos no afectan los cálculos por estar al cuadrado. Los valores del Gaussiano, en este caso, se pueden tomar de la sección 9.6.8, ya que habían sido calculados previamente. Véase la Tabla 9.1.



Figura 9.22: Funcionamiento principal del filtro Laplaciano del Gaussiano con ventana de 7×7 . (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Laplaciano del Gaussiano. Es adecuado para encontrar el borde en las imágenes, al ya que suaviza el ruido antes de buscar los contornos.

Posición (x,y)	$G_u(x, y)$	$\left[\frac{X^2+Y^2-2\sigma^2}{\sigma^4} \right]$	$G(x, y) \left[\frac{X^2+Y^2-2\sigma^2}{\sigma^4} \right]$	$\nabla_n^2 k(x, y)$
(0,0)	0.6187	$\frac{0^2+0^2-0.5}{0.0625} = -8$	$0.6187 \times -8 = -4.9496$	-4.9048
(0,1)	0.0837	$\frac{0^2+1^2-0.5}{0.0625} = 8$	$0.0837 \times 8 = 0.6696$	0.7146
(1,1)	0.0113	$\frac{1^2+1^2-0.5}{0.0625} = 24$	$0.0113 \times 24 = 0.2712$	0.3167
(0,2)	0.0002	$\frac{0^2+2^2-0.5}{0.0625} = 56$	$0.0002 \times 56 = 0.0112$	0.0564
(1,2)	0.0000	$\frac{1^2+2^2-0.5}{0.0625} = 72$	$0.0000 \times 72 = 0.0000$	0.0468
(2,2)	0.0000	$\frac{2^2+2^2-0.5}{0.0625} = 120$	$0.0000 \times 120 = 0.0000$	0.0448

Cuadro 9.3: Cálculos para las posiciones únicas en el cuadrante inferior derecho de la matriz 5×5 con desviación estándar de 0.5 para el LoG.

Finalmente, el resultado es el *LoG* normalizado es:

$$\nabla_n^2 k(x, y) = \begin{bmatrix} 0.0448 & 0.0468 & 0.0564 & 0.0468 & 0.0448 \\ 0.0468 & 0.3167 & 0.7146 & 0.3167 & 0.0468 \\ 0.0564 & 0.7146 & -4.9048 & 0.7146 & 0.0564 \\ 0.0468 & 0.3167 & 0.7146 & 0.3167 & 0.0468 \\ 0.0448 & 0.0468 & 0.0564 & 0.0468 & 0.0448 \end{bmatrix}$$

9.7.4. Filtro Sharpen

El filtro *sharpen* (o de realce) es una herramienta en procesamiento de imágenes diseñada para mejorar los detalles y resaltar los bordes de una imagen. Su fundamento se realiza con base en el operador Laplaciano, que responde a cambios rápidos en la intensidad de los píxeles. Este filtro permite realzar bordes y detalles, manteniendo la estructura de la imagen original. Formalmente, el kernel se define como:

$$K_{\text{sharpen}} = I - \nabla^2 I(x, y)_4$$

donde:

- **Matriz Identidad (I):** Se representa como una matriz de 3×3 en la que solo el elemento central es 1, mientras que el resto de los elementos son ceros, lo que permite que el filtro preserve la intensidad del píxel

central en la operación de convolución.

$$I = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- **Operador Laplaciano ($\nabla^2 I(x, y)_4$)**: Detecta variaciones rápidas de intensidad, asignando valores negativos a los píxeles circundantes al píxel central. Recordando, la forma general del Laplaciano ajustado de cuatro puntos es:

$$\nabla^2 I(x, y)_4 \approx \frac{1}{4(\alpha + 1)} \begin{bmatrix} \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \\ \frac{1-\alpha}{4} & -1 & \frac{1-\alpha}{4} \\ \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \end{bmatrix}$$

El parámetro α controla el peso de los píxeles circundantes en el Laplaciano. Cuando $\alpha = 0$, el operador Laplaciano se simplifica a:

$$\nabla^2 I(x, y)_4 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Por lo tanto, al usar la expresión $K_{\text{sharpen}} = I - \nabla^2 I(x, y)_4$, se obtiene el kernel de *sharpen* con $\alpha = 0$:

$$K_{\text{sharpen}} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

El kernel *sharpen* se caracteriza por su valor central intensificado, el cual, cuando $\alpha = 0$, alcanza un valor de 5, lo que permite destacar la intensidad del píxel central respecto a sus vecinos y acentuar los bordes y detalles de la imagen. Los valores negativos asignados a los píxeles circundantes potencian la detección de cambios abruptos de intensidad, contribuyendo a un efecto de definición en los bordes. En la Fig. 9.23 se aprecia el efecto de aplicar el filtro *Sharpen* sobre una imagen. En esta se resaltan los detalles.



Figura 9.23: Funcionamiento principal del filtro Sharpen (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Sharp. Se usa para resaltar los bordes de los objetos en imágenes con falta de detalles.

9.7.5. Filtro Sobel

Sobel es un operador de derivación discreta que se utiliza para calcular el gradiente de una imagen. Fue propuesto por el ingeniero electricista norteamericano Irwin Sobel en 1968 [5, 13, 52]. Este consiste en dos núcleos de convolución, uno para la dirección horizontal y otro para la dirección vertical

Deducción del Filtro Sobel

Se considera una ventana de 3×3 píxeles:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Se calcula la derivada discreta en x y y usando diferencias finitas centradas:

$$\frac{\partial f}{\partial x} \approx (c - a) + 2(f - d) + (i - g) \quad S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{\partial f}{\partial y} \approx (g - a) + 2(h - b) + (i - c) \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

La magnitud y el ángulo del gradiente se obtienen combinando los resultados de S_x y S_y :

$$S = \sqrt{S_x^2 + S_y^2}$$

$$\theta = \tan^{-1} \left(\frac{S_y}{S_x} \right)$$

s_x y s_y se pueden implementar usando los kernels de convolución que se muestran en (9.36):

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (9.36)$$

Este filtro es sencillo de implementar y eficiente desde el punto de vista computacional. Muestra buenos resultados al detectar bordes suaves y curvos, pero es muy sensible al ruido y a los cambios bruscos de dirección. Además, su tamaño fijo limita su capacidad para detectar bordes de diferentes tamaños en las imágenes. En las Figuras 9.24 y 9.25 se muestra un ejemplo de uso del filtro de Sobel en X y en Y, respectivamente.

9.7.6. Filtro Prewitt

Este es muy similar al operador Sobel, puesto que también detecta los bordes verticales y horizontales de una imagen. Fue propuesto por el científico de la computación británico Colin Prewitt (3 de diciembre de 1930, 23 de marzo de 2011) en 1970 en su artículo titulado "Object enhancement and extraction" en el cual describió su aporte [5, 13, 42]. El operador trabaja a través de la convolución con dos kernels de filtro que definen la primera derivada de la imagen como se muestran en (9.37).



Figura 9.24: Funcionamiento principal del filtro Sobel en X. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Sobel en la dirección X. Es adecuado para encontrar líneas horizontales y verticales en las imágenes.

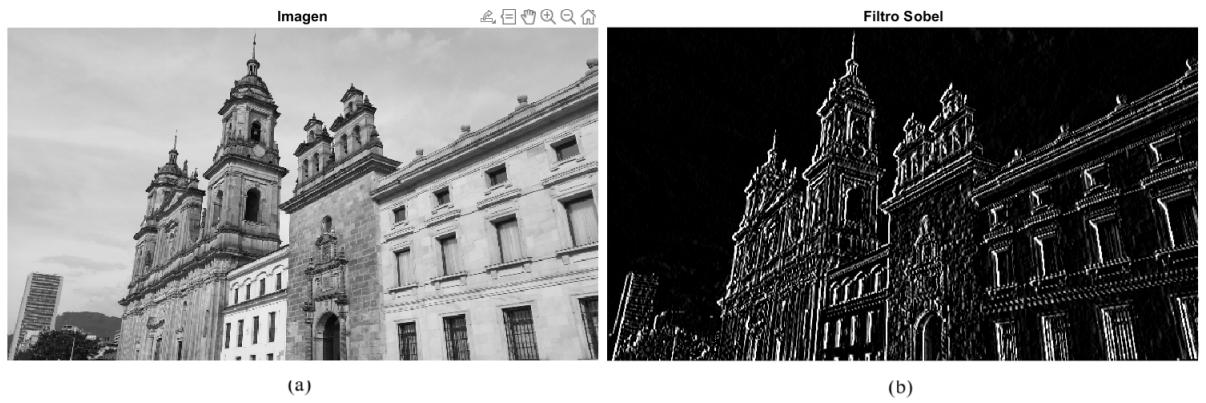


Figura 9.25: Funcionamiento principal del filtro Sobel en Y. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Sobel en la dirección Y. Es adecuado para encontrar líneas horizontales y verticales en las imágenes.

Deducción del Filtro Prewitt

Considere, nuevamente, una ventana de 3×3 píxeles:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Se calcula la derivada discreta en x y en y usando diferencias finitas centradas:

$$\frac{\partial f}{\partial x} \approx (c - a) + (f - d) + (i - g) \quad P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{\partial f}{\partial y} \approx (g - a) + (h - b) + (i - c) \quad P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

La magnitud y el ángulo del gradiente se obtienen combinando los resultados de P_x y P_y :

$$P = \sqrt{P_x^2 + P_y^2}$$

$$\theta = \tan^{-1} \left(\frac{P_y}{P_x} \right)$$

Al igual que los otros operadores de gradiente, P_x y P_y se pueden implementar usando kernels de convolución que se muestran en (9.37):

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}; \quad P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad (9.37)$$

El operador Prewitt es conocido por ser simple de implementar y eficiente computacionalmente. Es menos susceptible al ruido que el operador Sobel, pero puede tener dificultades para detectar bordes curvos y puede dar resultados menos precisos en gradientes de intensidad. Además, su tamaño fijo limita los tamaños de bordes que puede detectar en una imagen.

La razón por la cual no se multiplica por 2 el término central en los operadores Prewitt (como se hace en Sobel) radica en la interpretación de la derivada discreta centrada. Al no hacerlo, se logra que sea más suave en comparación con Sobel, lo que resulta beneficioso en imágenes con ruido. En las Figuras 9.26 y 9.27 se muestra un ejemplo de uso del filtro de Prewitt en X y en Y, respectivamente.



Figura 9.26: Funcionamiento principal del filtro Prewitt en X. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Prewitt en la dirección X. Es adecuado para encontrar líneas horizontales y verticales en las imágenes.

9.7.7. Ejemplo en Python: Filtros pasa bajo

Este código implementa diversos filtros pasa altos para el procesamiento de imágenes. Los operadores de realce como Prewitt, Sobel y Laplaciano se aplican para destacar los bordes y acentuar los detalles de alta frecuencia presentes en la imagen original. La presentación de resultados se organiza mostrando tres imágenes por fila, facilitando así la comparación directa entre los diferentes efectos obtenidos con cada filtro.

Solución

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from ip_functions import * # Asegúrate de que esta biblioteca esté disponible
4
5 # Cargar imagen
6 I = plt.imread("bogota.jpg")

```



Figura 9.27: Funcionamiento principal del filtro Prewitt en Y. (a) La imagen I sin ruido dominante. (b) La imagen I' con filtrado Prewitt en la dirección Y. Es adecuado para encontrar líneas horizontales y verticales en las imágenes.

```

7  I = np.array(I)
8  S = 4 # Factor de escala
9  I = I[0::S, 0::S, :]
10
11 # Definir filtros pasa altos
12 high_pass_filters = {
13     "Prewitt X": fspecial('prewitt'),
14     "Prewitt Y": fspecial('prewitt').T,
15     "Sobel X": fspecial('sobel'),
16     "Sobel Y": fspecial('sobel').T,
17     "LoG": fspecial('log')
18 }
19
20 # Aplicar filtros pasa altos
21 filtered_images = [
22     name: imfilter(I, k, 'same', 'symmetric')
23     for name, k in high_pass_filters.items()
24 ]
25
26 # Crear figura y mostrar imágenes en un solo subplot
27 num_filters = len(filtered_images)
28 cols = 3 # Tres columnas
29 # Filas necesarias
30 rows = (num_filters // cols) + (num_filters % cols > 0)
31
32 fig, axes = plt.subplots(
33     rows, cols,
34     figsize=(12, 6)
35 )
36 axes = axes.ravel()
37
38 # Mostrar la imagen original en la primera posición
39 axes[0].imshow(I, cmap='gray')
40 axes[0].set_title("(a) Original")
41 axes[0].axis('off')
42
43 # Mostrar imágenes filtradas
44 for i, (name, img) in enumerate(filtered_images.items(), start=1):
45     label =
46         "(b)" if i == 1 else
47         "(c)" if i == 2 else
48         "(d)" if i == 3 else
49         "(e)" if i == 4 else
50         "(f)"
51
52     axes[i].imshow(img, cmap='gray')
53     axes[i].set_title(f"{label} {name}")
54     axes[i].axis('off')
55

```

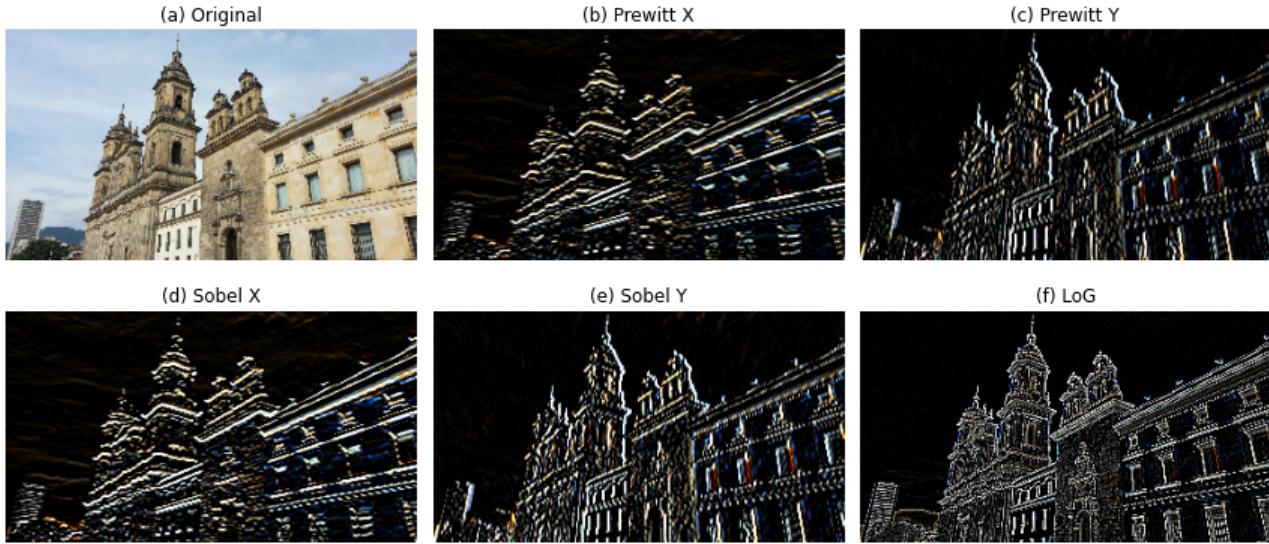


Figura 9.28: Comparación de filtros pasa altos aplicados a una imagen de Bogotá. (a) Imagen original; (b-c) Filtros Prewitt en direcciones X e Y que detectan bordes verticales y horizontales; (d-e) Filtros Sobel en direcciones X e Y con mayor robustez ante ruido; (f) Filtro Laplaciano del Gaussiano (LoG) que resalta contornos en todas las direcciones.

```

56 # Ocultar subplots vacíos si hay
57 for j in range(i + 1, rows * cols):
58     axes[j].axis('off')
59
60 plt.suptitle(
61     "Fig. 2: (a) Imagen original, (b) Prewitt X, (c) Prewitt Y, "
62     "(d) Sobel X, (e) Sobel Y, (f) LoG",
63     fontsize=12
64 )
65 plt.tight_layout()
66 plt.show()

```

La Fig. 9.28 presenta los resultados de aplicar diversos filtros pasa altos a una imagen de Bogotá submuestreada por un factor de 4, donde cada panel ilustra un efecto específico de realce: el panel (a) ilustra la imagen original sin modificaciones como referencia visual, mientras que los paneles (b) y (c) exhiben los filtros Prewitt en direcciones X e Y que detectan bordes verticales y horizontales respectivamente; los paneles (d) y (e) presentan los filtros Sobel en ambas direcciones, similares a los Prewitt pero con mayor peso central que reduce la sensibilidad al ruido; finalmente, el panel (f) muestra el filtro Laplaciano del Gaussiano (LoG) que detecta transiciones rápidas de intensidad en todas las direcciones, resaltando eficazmente contornos y esquinas mediante la combinación de un suavizado gaussiano con un operador laplaciano. Las funciones referidas allí, se encuentran en el capítulo 13.

Funciones

Las siguientes tienen relación con el tema abordado en este capítulo:

1. **fspecial(tipo, parámetros)**: crea un filtro espacial 2D del tipo especificado. El parámetro **tipo** define el filtro a generar, como '**'gaussian'**', '**'sobel'**', '**'prewitt'**', '**'laplacian'**', entre otros. Los parámetros adicionales dependen del tipo, como el tamaño del filtro y la desviación estándar en el caso del filtro gaussiano. Esta función genera una matriz que puede ser usada para convolución con imágenes.
2. **imfilter(I, h, opciones)**: aplica un filtro espacial 2D a una imagen **I**, utilizando el filtro **h** generado por **fspecial** u otra matriz definida. Esta función realiza convolución (o correlación) entre la imagen y el filtro, permitiendo suavizar, detectar bordes o realzar detalles. Las **opciones** pueden especificar cómo manejar los bordes ('**replicate**', '**symmetric**', etc.) y el modo de operación ('**conv**' o '**corr**').

3. `ordfilt2(I, orden, vecindad)`: realiza filtrado por orden sobre la imagen I . El parámetro `orden` indica qué posición del conjunto de valores ordenados se tomará en cada vecindad (por ejemplo, 1 para mínimo, 9 para máximo si la vecindad tiene 9 elementos). La `vecindad` define el patrón de vecindad (por ejemplo, una matriz de unos). Esta función permite construir filtros como el mínimo, máximo o mediana.
4. `medfilt2(I, tamaño)`: aplica un filtro de mediana bidimensional sobre la imagen I . El parámetro `tamaño` define la vecindad cuadrada (por defecto 3×3) sobre la que se calcula la mediana. Es especialmente efectivo para eliminar ruido tipo sal y pimienta, preservando bordes importantes de la imagen.
5. `modefilt(I, vecindad)`: realiza un filtrado por moda, reemplazando cada píxel de la imagen I por el valor más frecuente dentro de la vecindad especificada. El parámetro `vecindad` es una matriz binaria que define qué píxeles se consideran alrededor del píxel central. Este filtro es útil para eliminar ruido en imágenes categóricas o con pocos niveles de intensidad.
6. `stdfilt(I, vecindad)`: calcula la desviación estándar local para cada píxel de la imagen I , evaluando la variabilidad de los valores dentro de una `vecindad` definida (por defecto, 3×3). Las regiones con alta variación (bordes, texturas) presentan valores mayores, lo que permite detectar estructuras relevantes en la imagen.
7. `entropyfilt(I, vecindad)`: calcula la entropía local de la imagen.

9.8. Preguntas

9.8.1. Preguntas sobre hechos indicados en el texto

1. ¿Qué es el filtrado espacial en el procesamiento de imágenes?
2. ¿Cuál es la diferencia principal entre la correlación y la convolución en dos dimensiones?
3. ¿Cuál es la principal diferencia entre el operador Laplaciano y otros operadores de detección de bordes como Prewitt y Sobel?
4. ¿Qué es el filtro Laplacian of Gaussian (LoG) y para qué se utiliza?
5. ¿Cuál es la ventaja de usar el filtro Gaussiano ?

9.8.2. Preguntas de análisis y comprensión con base en el texto

1. Analice cómo la elección del tamaño de la ventana de un filtro de suavizado afecta el resultado final en una imagen.
2. Compare las propiedades y aplicaciones de los filtros de Sobel y Prewitt en la detección de bordes.
3. Explique por qué es importante la normalización en el uso del kernel Gaussiano.
4. Evalúe los efectos de aplicar un filtro de mediana en una imagen con ruido de "sal y pimienta".
5. Analice las diferencias entre el uso de un filtro Laplaciano de 4 puntos y uno de 8 puntos.

9.8.3. Ejercicios Numéricos

1. Calcule la correlación de una imagen I de 3×3 con un kernel k de 3×3 con los valores mostrados en el punto $I(1,1) = a$ utilizando padding "Simétrico".

$$I = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad k = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

2. Determine el valor filtrado de un píxel $I(2,2)$ y $I(3,3)$ utilizando un mediana con un kernel k de 3×3 . Use padding de "Symmetric".

$$I = \begin{bmatrix} 4 & 2 & 5 \\ 6 & 3 & 4 \\ 3 & 7 & 4 \end{bmatrix}$$

3. Calcule la convolución para el punto $I(1,1) = a$ de una imagen I de 3×3 con un kernel k de 3×3 utilizando padding "Circular".

$$I = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad k = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

4. Utilice el operador Laplaciano de 4 puntos para detectar bordes en una imagen I de 3×3 utilizando padding "Replicate".

$$I = \begin{bmatrix} 1 & 4 & 5 \\ 6 & 5 & 4 \\ 3 & 7 & 2 \end{bmatrix}$$

5. Calcule el Kernel Laplaciano de la Gaussiana (LoG) para una desviación estándar $\sigma = 0.8$.

Capítulo 10

Transformaciones Espaciales

Las transformaciones geométricas modifican las relaciones espaciales entre los píxeles de una imagen sin necesariamente alterar las intensidades de color. Estas son fundamentales en la visión por computadora, ya que permiten manipular la disposición de los objetos en una imagen mediante técnicas como traslaciones, rotaciones, escalamientos y cizallamientos, entre otras. Para lograrlo, se utilizan para extender el espacio euclíadiano, lo que facilita la representación de transformaciones complejas como rotaciones, traslaciones y escalados en una única matriz. Estas son fundamentales en la visión por computadora, ya que permiten corregir distorsiones capturadas desde diferentes ángulos, mejorar la calidad de imágenes y videos eliminando ruido, facilitar el seguimiento de objetos en secuencias visuales, y superponer objetos virtuales en escenas reales, como en la realidad aumentada [13,15,21,54].

Considérese una imagen $I(x, y)$, que experimenta una distorsión geométrica para generar una nueva imagen $I'(x', y')$, donde las combinaciones polinomiales generales de primer grado de x y y son:

$$x' = h_{11}x + h_{12}y + h_{13}, \quad y' = h_{21}x + h_{22}y + h_{23}$$

Aquí, $h_{11}, h_{12}, h_{13}, h_{21}, h_{22}$ y h_{23} son coeficientes que determinan cómo se transforman las coordenadas de la imagen original. El sistema se puede expresar de forma más compacta utilizando matrices con coordenadas homogéneas, así:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

La relación entre las coordenadas de la imagen original y la transformada se expresa como se ve en 10.1:

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \tag{10.1}$$

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

La matriz de transformación homogénea \mathbf{H} , de tamaño 3×3 , actúa sobre el vector de coordenadas homogéneas $\mathbf{p} = [x, y, 1]^T$ para generar el vector $\mathbf{p}' = [x', y', 1]^T$.

El uso de estas simplifica el tratamiento matemático de las distorsiones geométricas que se presentan frecuentemente en las imágenes capturadas por cámaras debido a la perspectiva. Además, facilita la representación de las transformaciones geométricas de manera matricial, permitiendo la aplicación de múltiples transformaciones consecutivas, tales como el escalamiento, la rotación y la translación, mediante productos de matrices. También da lugar a que la matriz sea invertible, lo cual es esencial para revertir el proceso de transformación cuando es necesario.

En las transformaciones afines, las cuales se describirán más adelante, siempre los términos $h_{31} = 0$ y $h_{32} = 0$ puesto que eliminan cualquier efecto proyectivo, lo que garantiza que la transformación se mantenga lineal en el plano 2D, mientras que $h_{33} = 1$ garantiza que $w = 1$, evitando la necesidad de normalizar las coordenadas, preservando las propiedades del espacio afín. En Fig.10.1 se muestran las transformaciones más comunes.

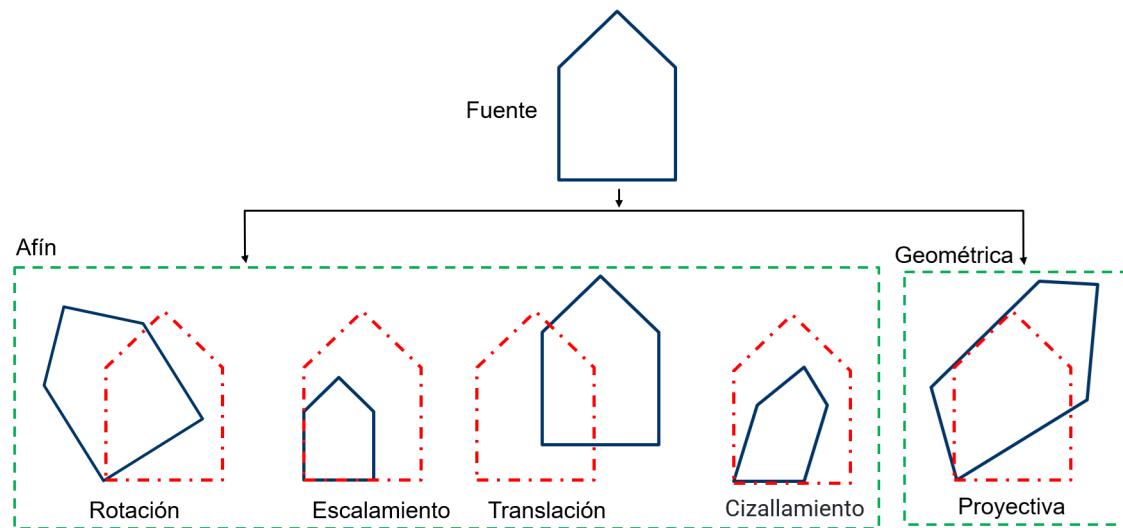


Figura 10.1: Se aprecian algunas transformaciones geométricas afines y proyectivas, incluyendo la translación, el escalamiento, la rotación, el cizallamiento y la transformación proyectiva de cuatro puntos.

10.1. Método de Transformación Directo

El método directo para determinar la imagen de salida en una transformación consiste en recorrer cada píxel \mathbf{p} de la imagen original $I(x, y)$ y calcular la posición donde debería ubicarse el nuevo punto \mathbf{p}' de la imagen de salida $I'(x', y')$, asignándole la intensidad correspondiente $I'(x', y') = I(x, y)$. Si bien este enfoque puede ser intuitivo, el proceso de redondeo y otras aproximaciones inherentes a los cálculos con números enteros pueden generar artefactos, al mismo tiempo que se puede producir que algunas posiciones no reciban valor alguno, pese a que deberían tenerlo. Para mitigar estos problemas, se utilizan técnicas de interpolación y otros enfoques más robustos que distribuyen las intensidades de manera más adecuada. Véase la Fig.10.2.

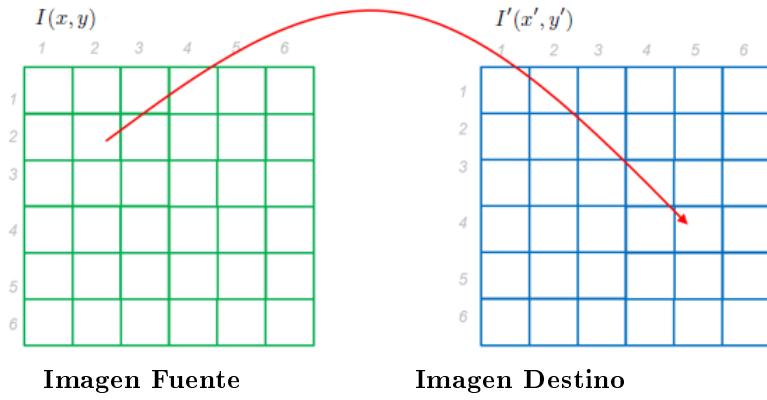


Figura 10.2: Transformación directa para determinar la nueva posición en las transformaciones geométricas. En este caso se recorre la imagen fuente I en la búsqueda de la posición donde se reubicará la intensidad en la imagen destino I' .

Se utiliza cuando se tienen los puntos originales $\mathbf{p} = [x, y, 1]^T$, los cuales son transformados a los puntos $\mathbf{p}' = [x', y', 1]^T$ mediante la multiplicación por la matriz H , así:

$$\mathbf{p}' = H\mathbf{p}$$

De forma matricial, esto se expresa como:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Después de realizar la multiplicación, se obtienen las ecuaciones para las coordenadas transformadas x' y y' :

$$x' = h_{11}x + h_{12}y + h_{13} \quad y' = h_{21}x + h_{22}y + h_{23}$$

En las transformaciones afines, los $h_{31} = 0$ y $h_{32} = 0$ eliminan cualquier efecto proyectivo, lo que asegura que esta se mantenga lineal en el plano 2D, mientras que $h_{33} = 1$ garantiza que $w = 1$, evitando la necesidad de normalizar las coordenadas y preservando las propiedades del espacio.

10.2. Método de Transformación Inverso

El método inverso de transformación consiste en recorrer cada píxel \mathbf{p}' de la imagen de salida $I'(x', y')$ y aplicar el mapeo inverso para encontrar la posición \mathbf{p} correspondiente en la imagen original $I(x, y)$. A diferencia del método directo, este enfoque garantiza que no queden sin asignar intensidades en la imagen de salida, para lo cual se recorre el destino y se busca el valor adecuado en la fuente. Esta técnica ofrece una asignación completa de valores, pero depende de la calidad de una posible interpolación para evitar la aparición de artefactos en la imagen destino. Véase la Fig. 10.3.

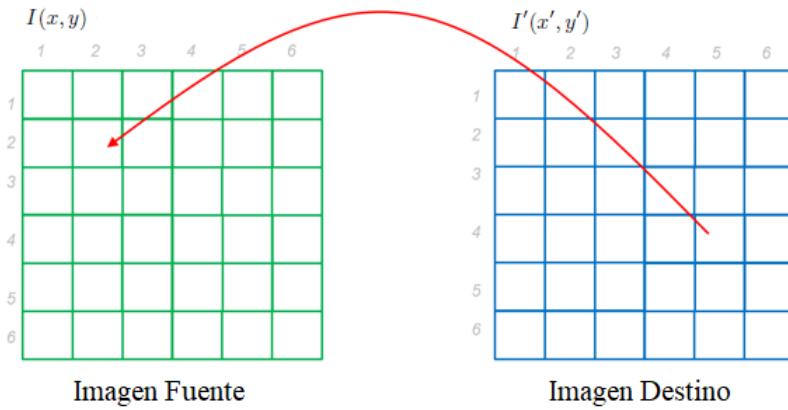


Figura 10.3: Transformación inversa para determinar la nueva posición en las transformaciones geométricas. En este caso se recorre la imagen destino I' en la búsqueda de la posición en la imagen fuente I donde está la intensidad que le corresponde.

El método inverso se utiliza cuando se tienen los puntos proyectados $\mathbf{p}' = [x', y', w]^T$ y se desea recuperar los puntos originales $\mathbf{p} = [x, y, 1]^T$, lo cual se logra mediante la matriz inversa H^{-1} , que “deshace” la transformación inicial:

$$\mathbf{p} = H^{-1}\mathbf{p}'$$

De forma matricial, el proceso es:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h'_{11} & h'_{12} & h'_{13} \\ h'_{21} & h'_{22} & h'_{23} \\ h'_{31} & h'_{32} & h'_{33} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w \end{bmatrix}$$

Finalmente, las coordenadas originales $\mathbf{p} = [x, y, 1]^T$ se obtienen dividiendo por w :

$$x = \frac{h'_{11}x' + h'_{12}y' + h'_{13}}{h'_{31}x' + h'_{32}y' + h'_{33}}, \quad y = \frac{h'_{21}x' + h'_{22}y' + h'_{23}}{h'_{31}x' + h'_{32}y' + h'_{33}}$$

Para las transformaciones afines, los valores son $h'_{31} = 0$, $h'_{32} = 0$, $h'_{33} = 1$ y $w = 1$. Esta configuración asegura que el sistema se mantenga lineal, evitando la introducción de efectos de proyección y así garantizando que las coordenadas cambian de manera consistente en el plano 2D.

10.3. Transformaciones en Sistema Cartesiano y Matricial

Al aplicar transformaciones geométricas a imágenes digitales, es fundamental comprender las diferencias entre el sistema de coordenadas cartesianas y el matricial. Aunque ambos sistemas se utilizan para representar puntos en un espacio bidimensional, su notación y el enfoque afectan cómo se interpretan en el procesamiento digital de imágenes.

10.3.1. Sistema de Coordenadas Cartesiano

Un punto P se define mediante un par ordenado de coordenadas $(x, y) \in \mathbb{R}^2$, donde:

- x representa la posición sobre el eje horizontal (o eje x).
- y representa la posición sobre el eje vertical (o eje y).

Este sistema se emplea ampliamente en geometría y gráficos, con el eje x extendiéndose convencionalmente de izquierda a derecha y el eje y de abajo hacia arriba.

10.3.2. Sistema Matricial

El sistema matricial organiza las posiciones en un arreglo bidimensional $I \in \mathbb{R}^{M \times N}$, en el cual:

- M representa el número de filas (la altura de la imagen).
- N representa el número de columnas (el ancho de la imagen).

Cada elemento en esta matriz se identifica mediante un par de índices que corresponden a una fila y una columna, donde:

- La fila corresponde a la posición vertical de un píxel en la imagen, con valores de 1 a M .
- La columna corresponde a la posición horizontal de un píxel en la imagen, con valores de 1 a N .

10.3.3. Interpretación de Coordenadas

Al aplicar una transformación geométrica en el sistema cartesiano, el punto transformado (x', y') representa las nuevas coordenadas en el plano cartesiano. Para adaptarlas al sistema matricial, se realiza la siguiente asignación:

$$X' = y' \quad \text{y} \quad Y' = x'$$

De este modo, las coordenadas transformadas se interpretan en la matriz como (X', Y') , donde:

- X' corresponde a la fila, que en el sistema cartesiano es la componente y' (vertical).
- Y' corresponde a la columna, que en el sistema cartesiano es la componente x' (horizontal).

10.3.4. Aplicación de Transformaciones a Imágenes

Cuando se aplica una transformación geométrica $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, el resultado se da en el sistema cartesiano como (x', y') . Para convertir este punto al sistema matricial, se asigna así:

$$I(X, Y) = I(y, x) \quad I'(X', Y') = I(y', x')$$

Ejemplo:

Si en el sistema cartesiano un punto es $P(3, 2)$, en el sistema matricial se representará como $I(2, 3)$.

10.4. Transformaciones Geométricas Afines

Una transformación afín es una operación matemática que aplica a un conjunto de puntos, preservando la colinealidad y el paralelismo entre las líneas. En otras palabras, si tres puntos estaban alineados en la imagen original $I(x, y)$, seguirán estando de igual forma tras la transformación en la imagen destino $I'(x', y')$. Aunque las distancias relativas y los ángulos entre las líneas pueden no conservarse, las proporciones y el paralelismo entre segmentos sí se mantienen. Esto es especialmente útil en la visión por computadora cuando se busca corregir distorsiones de perspectiva o ajustar la escala y la posición de un objeto en una escena, sin alterar sus relaciones espaciales fundamentales. Dentro las transformaciones afines está: la translación, el escalamiento, la rotación y el cizallamiento [13].

10.4.1. Translación

La translación realiza la asignación de la posición \mathbf{p} de cada elemento de la imagen de entrada $I(x, y)$ a una nueva \mathbf{p}' en una imagen de salida $I'(x', y')$ sin alterar el tamaño real de la imagen más allá de los límites de la translación que permiten que la imagen no esté fuera de su marco, cuando se requiere. La translación se utiliza para relocalizar la visualización de una imagen con respecto a una nueva referencia, por lo que se usa con frecuencia para la superposición de dos o más imágenes. En la Fig.10.4 se ilustra la translación de una forma (línea continua) a una posición final (línea discontinua) dentro de una escena [13,15].

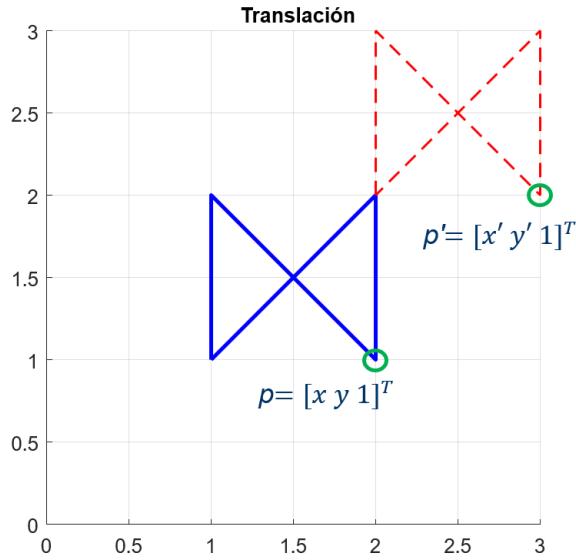


Figura 10.4: Representación de la translación de una forma (línea continua) a una posición final (línea discontinua) dentro de una escena. Nótese que, aunque los puntos cambian de posición, se mantiene el paralelismo final.

En el contexto de las transformaciones geométricas, cada píxel de una imagen fuente I con coordenadas $\mathbf{p} = [x, y, 1]^T$ se traslada a una nueva posición en la imagen destino I' , representada por $\mathbf{p}' = [x', y', 1]^T$. Esta operación de translación puede describirse añadiendo desplazamientos t_x y t_y a las coordenadas originales x y y , lo cual puede expresarse mediante las siguientes ecuaciones:

$$x' = x + t_x \quad y' = y + t_y$$

Para representar de manera más conveniente esta transformación, se utiliza la notación matricial en coordenadas homogéneas, donde tanto \mathbf{p} como \mathbf{p}' son vectores homogéneos y \mathbf{T} es una matriz de translación, por lo que el sistema puede escribirse como:

$$\mathbf{p}' = \mathbf{T}\mathbf{p}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

donde:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

La Figura 10.5 se muestra un ejemplo de translación de una imagen. Nótese, en este caso, que se ha dejado de un tamaño más grande para no recortar la translación.



Figura 10.5: A la izquierda: imagen original tomada en el Museo Botero de la ciudad de Bogotá de sus obras “La Carta” y “Naranjas”. A la derecha: ejemplo de imagen transladada.

10.4.2. Algoritmo Inverso de Translación

El proceso de translación de una imagen implica relocalizar los píxeles de la imagen original mediante un desplazamiento en las direcciones x y y . Sea I la imagen original con dimensiones $M \times N$, donde cada píxel está en un punto de coordenadas homogéneas:

$$\mathbf{p} = [x \quad y \quad 1]^T$$

Para determinar el tamaño de la nueva imagen transladada, se aplica la matriz de translación \mathbf{T} a las dimensiones originales. La nueva imagen I' tendrá dimensiones $M' \times N'$:

$$\mathbf{z}' = \mathbf{D}\mathbf{z}$$

donde:

$$\mathbf{z}' = \begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 1 & 0 & |t_x| \\ 0 & 1 & |t_y| \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix}$$

Nótese, que en este caso, la matriz \mathbf{D} es diferente \mathbf{T} en los valores absolutos en su interior. t_x y t_y , son los desplazamientos en las direcciones x y y , respectivamente los cuales pueden ser positivos o negativos. Es importante observar que en algunos casos se decide mantener las dimensiones de la imagen transladada sin cambios, por lo que solo se aplica la modificación a los píxeles dentro de las dimensiones originales $M \times N$.

Para cada píxel en la nueva imagen transladada I' , se calcula la posición correspondiente en la imagen original I . Las coordenadas homogéneas de un píxel en la imagen transladada se expresan como:

$$\mathbf{p}' = [x' \quad y' \quad 1]^T$$

Las coordenadas originales \mathbf{p} en la imagen I se obtienen aplicando la inversa de la matriz de translación \mathbf{T} :

$$\mathbf{p} = \mathbf{T}^{-1} \mathbf{p}'$$

donde:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

La inversa de la matriz \mathbf{T}^{-1} se obtiene cambiando los signos de los términos t_x y t_y , ya que la translación es una operación aditiva.

Seguidamente, las coordenadas originales (x, y) se redondean al píxel más cercano:

$$x = \text{round}(x) \quad y = \text{round}(y)$$

Finalmente, se verifica si estas coordenadas están dentro de los límites de las dimensiones de la imagen original I :

$$1 \leq x \leq N \quad 1 \leq y \leq M$$

Si las coordenadas (x, y) están dentro de estos límites, se asigna el valor del píxel de la imagen original $I(x, y)$ al píxel correspondiente en la imagen transladada $I'(x', y')$. Esto se escribe como:

$$I'(x', y') = I(x, y)$$

Este método de translación inversa asegura que cada píxel en la imagen transladada I' tenga un valor asignado, manteniendo la integridad en la región donde se superponen.

Ejemplo de Translación

Dada una imagen original I de tamaño 3×3 con la siguiente matriz de píxeles, se requiere aplicar una translación a la imagen utilizando los desplazamientos $t_x = 2$ en la dirección x y $t_y = 1$ en la dirección y . Se propone calcular la nueva posición de cada píxel en la imagen transladada I' de la matriz de salida, utilizando coordenadas homogéneas.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Solución

La imagen original I tiene las siguientes dimensiones:

$$M = 3 \quad (\text{filas}) \quad N = 3 \quad (\text{columnas})$$

Los desplazamientos en las direcciones x y y son:

$$t_x = 2 \quad t_y = 1$$

La matriz de dimensión \mathbf{D} es:

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & |t_x| \\ 0 & 1 & |t_y| \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Se multiplican la matriz \mathbf{D} por las dimensiones originales \mathbf{z} :

$$\mathbf{z}' = \mathbf{D}\mathbf{z}$$

Donde:

$$\mathbf{z}' = \begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} \quad \mathbf{z}' = \mathbf{D}\mathbf{z} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} \Rightarrow \mathbf{z}' = \begin{bmatrix} 5 \\ 4 \\ 1 \end{bmatrix}$$

Por lo tanto, las nuevas dimensiones de la imagen son:

$$M' = 4 \quad (\text{filas}) \quad N' = 5 \quad (\text{columnas})$$

Así, la matriz de translación T y su inversa \mathbf{T}^{-1} son respectivamente:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

Seguidamente, se crea una matriz vacía para la imagen transladada I' con las nuevas dimensiones 4×5 .

Para cada píxel en la nueva imagen I' , se calcula su posición en la imagen original I mediante la matriz inversa \mathbf{T}^{-1} .

Ejemplo 1: Para el punto $\mathbf{p}' = [4, 2, 1]^T$ (Cartesiano), se tiene:

$$\mathbf{p} = \mathbf{T}^{-1}\mathbf{p}' = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 4-2 \\ 2-1 \\ 1 \end{bmatrix} \Rightarrow \mathbf{p} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

Las coordenadas $\mathbf{p} = [2, 1, 1]^T$ (Cartesiana) están dentro de los límites del dominio de la imagen en los ejes x y y . Por lo tanto, al hacer la interpretación matricial correspondiente, se tiene que la intensidad en $I(1, 2) = 2$, lo que implica que, tras la transformación, la intensidad que se asigna a $I'(2, 4) = 2$.

Como se ha explicado anteriormente, fue necesario ajustar las coordenadas resultantes al formato matricial, por lo que se llevaron a índices matriciales (X, Y) , que relacionan las filas y columnas de la imagen usando la relación $I(X, Y) = I(y, x)$ y $I'(X', Y') = I(y', x')$.

Ejemplo 2: Para el píxel $\mathbf{p}' = [3, 3, 1]^T$, se aplica la inversa de la matriz de translación y se redondea $\lfloor \mathbf{p} \rfloor$:

$$\mathbf{p} = \mathbf{T}^{-1}\mathbf{p}' = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3-2 \\ 3-1 \\ 1 \end{bmatrix} \implies \mathbf{p} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

Las coordenadas $\mathbf{p} = [1, 2, 1]^T$ (Cartesiana) están dentro de los límites de la imagen en los ejes x y y . Haciendo la interpretación matricial, se tiene que la intensidad en $I(2, 1) = 4$, corresponde a la intensidad en la posición $I'(3, 3) = 4$ de la imagen transformada.

La Tabla 10.1 resume los resultados de todos los valores mapeados matricialmente para el algoritmo inverso de translación donde se presenta la posición en la matriz translada $(X', Y') = (y', x')$, la encontrada en la matriz fuente $(X, Y) = (y, x)$, la redondeada $\lfloor (y, x) \rfloor$ y finalmente la intensidad correspondiente $I'(X', Y') = I(X, Y)$.

(y', x')	(y, x)	$\lfloor (y, x) \rfloor$	$I'(y', x')$	(y', x')	(y, x)	$\lfloor (y, x) \rfloor$	$I'(y', x')$
(1,1)	(0.00,-1.00)	(0,-1)	0	(2,3)	(1.00,1.00)	(1,1)	1
(1,2)	(0.00,0.00)	(0,0)	0	(2,4)	(1.00,2.00)	(1,2)	2
(1,3)	(0.00,1.00)	(0,1)	0	(2,5)	(1.00,3.00)	(1,3)	3
(1,4)	(0.00,2.00)	(0,2)	0	(3,1)	(2.00,-1.00)	(2,-1)	0
(1,5)	(0.00,3.00)	(0,3)	0	(3,2)	(2.00,0.00)	(2,0)	0
(2,1)	(1.00,-1.00)	(1,-1)	0	(3,3)	(2.00,1.00)	(2,1)	4
(2,2)	(1.00,0.00)	(1,0)	0	(3,4)	(2.00,2.00)	(2,2)	5
(3,5)	(2.00,3.00)	(2,3)	6	(4,1)	(3.00,-1.00)	(3,-1)	0
(4,2)	(3.00,0.00)	(3,0)	0	(4,3)	(3.00,1.00)	(3,1)	7
(4,4)	(3.00,2.00)	(3,2)	8	(4,5)	(3.00,3.00)	(3,3)	9

Cuadro 10.1: Tabla de resultados de la translación. Se resumen los resultados de todos los valores mapeados matricialmente para el algoritmo inverso de translación donde se presenta la posición en la matriz translada (y', x') , la encontrada en la matriz fuente (y, x) , la redondeada $\lfloor (y, x) \rfloor$ y finalmente la intensidad correspondiente $I'(y', x') = I(y, x)$.

Aplicando este proceso para todos los píxeles, se obtiene la matriz para la imagen translada I' :

$$I' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 4 & 5 & 6 \\ 0 & 0 & 7 & 8 & 9 \end{bmatrix}$$

10.4.3. Escalamiento

La transformación geométrica de escalamiento se usa para reducir o ampliar el tamaño de una imagen I a una nueva I' de nuevas dimensiones,. Esto se logra mediante la relocalización entre valores de los píxeles de la vecindad original de la imagen. En la Fig.10.6 se ilustra el escalamiento de una forma (línea continua) a un tamaño final (línea discontinua) dentro de una escena [13, 15].

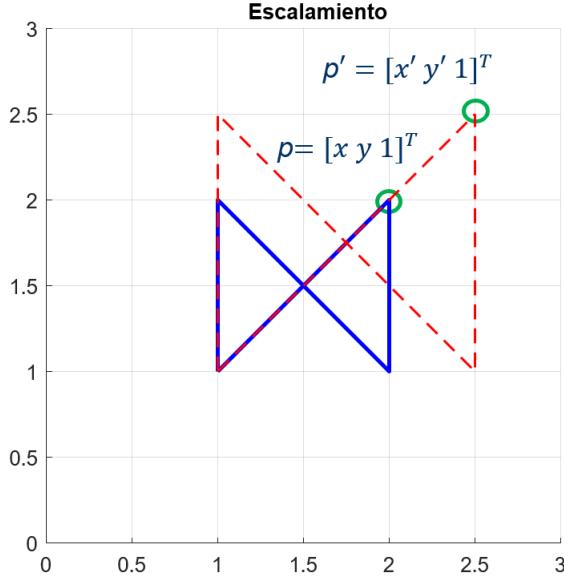


Figura 10.6: Representación del escalamiento de una forma (línea continua) a una tamaño final (línea discontinua) dentro de una escena. Nótese que, aunque las líneas cambian de longitud, se mantiene el paralelismo final.

En el proceso de escalamiento geométrico, cada píxel de la imagen fuente I con coordenadas $\mathbf{p} = [x, y, 1]^T$ se reasigna a una nueva posición en la imagen destino I' , $\mathbf{p}' = [x', y', 1]^T$, multiplicando cada una de las coordenadas iniciales por un factor de escala. Este proceso puede expresarse mediante las ecuaciones:

$$x' = S_x x \quad y' = S_y y$$

donde S_x y S_y son los factores de escala en las direcciones x y y , respectivamente. Para representar esta transformación de manera matricial, se utiliza la notación homogénea, en la que tanto \mathbf{p} como \mathbf{p}' son vectores homogéneos y \mathbf{S} es la matriz homogénea de escalamiento. Así, la transformación puede escribirse como:

$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$

donde:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplicando \mathbf{p} por \mathbf{S} , se obtiene \mathbf{p}' :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Este producto matricial genera las nuevas coordenadas directas escaladas de la imagen transformada. En la mayoría

de los casos, $S_x = S_y$ para mantener la proporcionalidad. No obstante, estos no tienen que ser iguales, lo que permite aplicar modificaciones no uniformes si es necesario.

La Figura 10.7 se muestra un ejemplo de escalamiento de una imagen. Nótese, en este caso, que el tamaño que la imagen escalada se ha reducido por lo que se ven sus píxeles al modificar su tamaño.

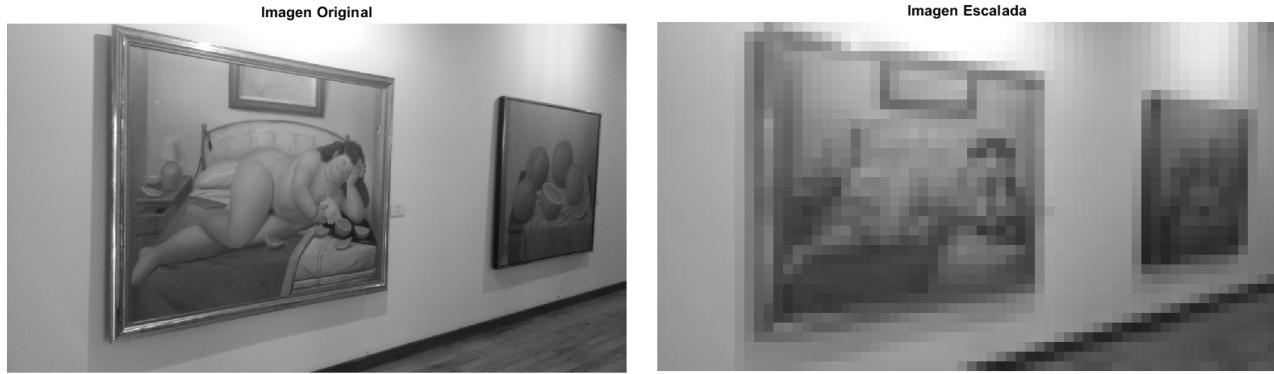


Figura 10.7: A la izquierda: imagen original tomada en el Museo Botero de la ciudad de Bogotá de sus obras “La Carta” y “Naranjas”. A la derecha: ejemplo de imagen escalada.

10.4.4. Algoritmo Inverso de Escalamiento

El proceso de escalamiento de una imagen implica ajustar las dimensiones de la imagen original mediante un factor de escala, para obtener una nueva imagen con dimensiones diferentes.

Sea I la imagen original con dimensiones $M \times N$, donde cada píxel de la imagen original $I(x, y)$ está en un punto de coordenadas homogéneas:

$$\mathbf{p} = [x \quad y \quad 1]^T$$

Para determinar el tamaño de la imagen escalada I' , se aplica la matriz de escalamiento a las dimensiones originales de la imagen I' . La nueva matriz tendrá dimensiones $N' \times M'$:

$$\mathbf{z}' = \mathbf{S}\mathbf{z}$$

donde:

$$\mathbf{z}' = \begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix}$$

S_x y S_y son los factores de escala en las direcciones x y y , respectivamente. Nótese, que en este caso, la matriz $\mathbf{D} = \mathbf{S}$.

Para cada píxel en la nueva imagen escalada I' , se calcula la posición correspondiente en la imagen original I . Las coordenadas homogéneas de un píxel en la imagen escalada se expresan como:

$$\mathbf{p}' = [x' \quad y' \quad 1]^T$$

Las coordenadas originales \mathbf{p} en la imagen I se obtienen aplicando la inversa de la matriz de escalamiento \mathbf{S} :

$$\mathbf{p} = \mathbf{S}^{-1}\mathbf{p}'$$

donde:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{S}^{-1} = \begin{bmatrix} S_x^{-1} & 0 & 0 \\ 0 & S_y^{-1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para encontrar la inversa de \mathbf{S} , se invierten los elementos de la diagonal principal, siempre que $S_x \neq 0$ y $S_y \neq 0$. Esto se debe a que la inversa de una matriz diagonal es otra matriz diagonal cuyos elementos son los inversos de los elementos originales.

Las coordenadas originales (x, y) se redondean al píxel más cercano:

$$x = \text{round}(x) \quad y = \text{round}(y)$$

Finalmente, se verifica si estas coordenadas están dentro de los límites de la imagen original I :

$$1 \leq x \leq M \quad 1 \leq y \leq N$$

Si las coordenadas (x, y) están dentro de estos límites, se asigna el valor del píxel de la imagen original $I(x, y)$ al píxel correspondiente en la imagen escalada $I'(x', y')$. Esto se escribe como:

$$I'(y', x') = I(x, y)$$

\text{''}

El método inverso asegura que cada píxel en la imagen escalada tenga un valor asignado, evitando así la aparición de .º agujeros.º píxeles sin información en la imagen resultante.

Ejemplo de Escalamiento

Se requiere realizar el proceso de **escalamiento** con factores de escala $S_x = S_y = 2$ para la matriz I :

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

El objetivo es ampliar la matriz original, duplicando su tamaño en ambas direcciones (ejes x y y) utilizando los factores de escala mostrados.

Solución

La imagen original I tiene las siguientes dimensiones:

$$M = 3 \quad (\text{filas}), \quad N = 3 \quad (\text{columnas})$$

Los escalamientos en las direcciones x y y son:

$$S_x = 2 \quad S_y = 2$$

La matriz de dimensión \mathbf{S} es:

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para determinar el tamaño de la imagen escalada I' , se aplica la matriz de escalamiento a las dimensiones originales $N = 3$ y $M = 3$ de la imagen I la cual tendrá las nuevas dimensiones $M' \times N'$. Esto se expresa mediante la transformación:

Se multiplican la matriz \mathbf{S} por las dimensiones originales \mathbf{z} :

$$\mathbf{z}' = \mathbf{S}\mathbf{z}$$

$$\mathbf{z}' = \begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} \quad \mathbf{z}' = \mathbf{S}\mathbf{z} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \\ 1 \end{bmatrix}$$

Por lo tanto, las nuevas dimensiones de la imagen I' son:

$$M' = 6 \quad (\text{filas}), \quad N' = 6 \quad (\text{columnas})$$

Así, la inversa de la matriz de escalamiento es \mathbf{S}^{-1} es:

$$\mathbf{S}^{-1} = \begin{bmatrix} S_x^{-1} & 0 & 0 \\ 0 & S_y^{-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Con esta información, se crea una matriz de ceros para la imagen escalada I' con las dimensiones 6×6 .

Para cada píxel en la nueva imagen I' , se calcula su posición en la imagen original I mediante la inversa \mathbf{S}^{-1} .

Ejemplo 1: Para las coordenadas escaladas $\mathbf{p}' = [1 \ 6 \ 1]^T$, se aplica la inversa de la matriz de escalamiento y se redondea $\lfloor \mathbf{p} \rfloor$:

$$\mathbf{p} = \mathbf{S}^{-1}\mathbf{p}' = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 6 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ 3 \\ 1 \end{bmatrix} \implies \mathbf{p} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

En el contexto de la imagen digital representada como una matriz, estas coordenadas se interpretan como la posición $I(3, 1)$, lo que significa que la intensidad almacenada en esta posición es $I(3, 1) = 7$, se reasigna en la

posición $I'(6, 1) = 7$, en la nueva imagen escalada.

Ejemplo 2: Para las coordenadas escaladas $\mathbf{p}' = [\begin{matrix} 6 & 2 & 1 \end{matrix}]^T$, nuevamente se aplica la inversa de la matriz de escalamiento y se redondea $\lfloor \mathbf{p} \rfloor$:

$$\mathbf{p} = \mathbf{S}^{-1}\mathbf{p}' = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \implies \mathbf{p} = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}$$

La intensidad en $I(1, 3) = 3$ se reemplaza en $I'(2, 6) = 3$ matricialmente.

De igual manera de procede para todo los valores de $I'(x', y')$.

Como se ha explicado anteriormente, fue necesario ajustar las coordenadas resultantes al formato matricial, por lo que se llevaron a índices, que relacionan las filas y columnas de la imagen usando la relaciones $I'(y', x') = I(y, x)$ para expresarlos adecuadamente de manera matricial.

La Tabla 10.2 resume los resultados de todos los valores mapeados matricialmente para el algoritmo inverso de escalamiento donde se presenta la posición en la matriz transladada (y', x') , la encontrada en la matriz fuente (y, x) , la redondeada $\lfloor (y, x) \rfloor$ y finalmente la intensidad correspondiente $I'(y', x') = I(y, x)$.

(y', x')	(y, x)	$\lfloor (y, x) \rfloor$	$I'(y', x)$	(y', x')	(y, x)	$\lfloor (y, x) \rfloor$	$I'(y', x)$
(1,1)	(0.50,0.50)	(1,1)	1	(1,2)	(0.50,1.00)	(1,1)	1
(1,3)	(0.50,1.50)	(1,2)	2	(1,4)	(0.50,2.00)	(1,2)	2
(1,5)	(0.50,2.50)	(1,3)	3	(1,6)	(0.50,3.00)	(1,3)	3
(2,1)	(1.00,0.50)	(1,1)	1	(2,2)	(1.00,1.00)	(1,1)	1
(2,3)	(1.00,1.50)	(1,2)	2	(2,4)	(1.00,2.00)	(1,2)	2
(2,5)	(1.00,2.50)	(1,3)	3	(2,6)	(1.00,3.00)	(1,3)	3
(3,1)	(1.50,0.50)	(2,1)	4	(3,2)	(1.50,1.00)	(2,1)	4
(3,3)	(1.50,1.50)	(2,2)	5	(3,4)	(1.50,2.00)	(2,2)	5
(3,5)	(1.50,2.50)	(2,3)	6	(3,6)	(1.50,3.00)	(2,3)	6
(4,1)	(2.00,0.50)	(2,1)	4	(4,2)	(2.00,1.00)	(2,1)	4
(4,3)	(2.00,1.50)	(2,2)	5	(4,4)	(2.00,2.00)	(2,2)	5
(4,5)	(2.00,2.50)	(2,3)	6	(4,6)	(2.00,3.00)	(2,3)	6
(5,1)	(2.50,0.50)	(3,1)	7	(5,2)	(2.50,1.00)	(3,1)	7
(5,3)	(2.50,1.50)	(3,2)	8	(5,4)	(2.50,2.00)	(3,2)	8
(5,5)	(2.50,2.50)	(3,3)	9	(5,6)	(2.50,3.00)	(3,3)	9
(6,1)	(3.00,0.50)	(3,1)	7	(6,2)	(3.00,1.00)	(3,1)	7
(6,3)	(3.00,1.50)	(3,2)	8	(6,4)	(3.00,2.00)	(3,2)	8
(6,5)	(3.00,2.50)	(3,3)	9	(6,6)	(3.00,3.00)	(3,3)	9

Cuadro 10.2: Tabla de resultados de la escalamiento. Se resumen los resultados de todos los valores mapeados matricialmente para el algoritmo inverso de escalamiento donde se presenta la posición en la matriz transladada (y', x') , la encontrada en la matriz fuente (y, x) , la redondeada $\lfloor (y, x) \rfloor$ y finalmente la intensidad correspondiente $I'(y', x') = I(y, x)$.

Finalmente, la matriz resultante $I'(\mathbf{x}', \mathbf{y}')$ es:

$$I' = \begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \\ 4 & 4 & 5 & 5 & 6 & 6 \\ 4 & 4 & 5 & 5 & 6 & 6 \\ 7 & 7 & 8 & 8 & 9 & 9 \\ 7 & 7 & 8 & 8 & 9 & 9 \end{bmatrix}$$

10.4.5. Rotación

En la rotación todos los puntos se mueven alrededor de un punto. Para esto, debe definirse un punto de rotación, que suele ser el origen de coordenadas en $(1,1)$, y el ángulo de giro. En este caso, no hay cambio de la escala de imagen rotada, por lo que se deben mantener las distancias entre los puntos. En la Fig.10.8 se ilustra la rotación de una forma (línea continua) a una posición final (línea discontinua) dentro de una escena [13,15].

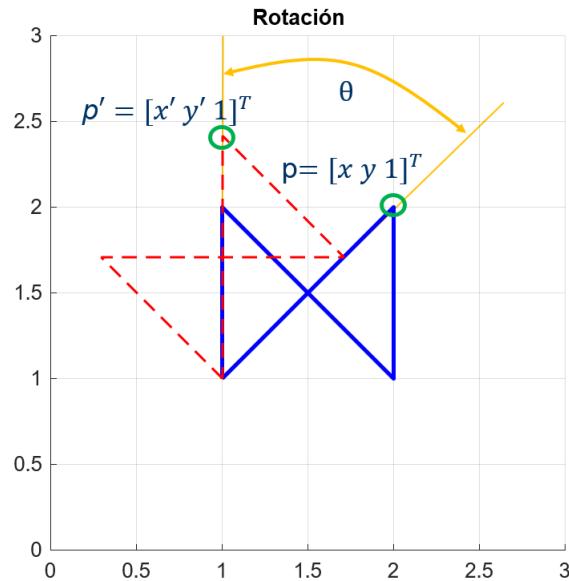


Figura 10.8: Representación de la rotación de una forma (línea continua) a una posición final (línea discontinua) al rededor de un punto dentro de una escena. Nótese que, las líneas no cambian de longitud y se mantiene el paralelismo final.

Los algoritmos de rotación son generalmente los más complejos y por lo tanto los más costosos en tiempo de procesado. Debido a esto, sólo se utilizan cuando es posible obtener una posición de giro que simplifique realmente los procesos posteriores. La determinación de las ecuaciones de giro se puede dar mediante la gráfica mostrada en la Fig.10.8:

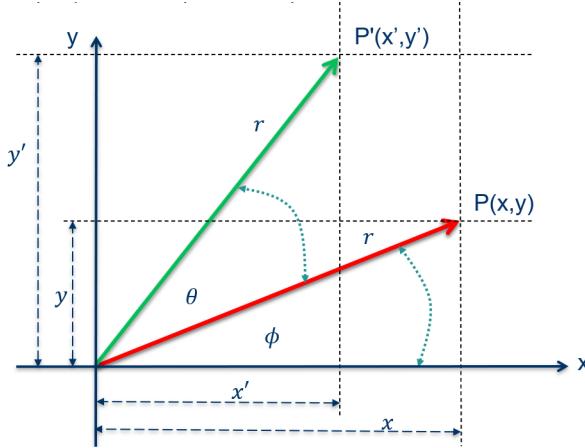


Figura 10.9: Representación geométrica para la obtención de los puntos de giro.

Para describir la rotación de un punto en el plano cartesiano, se considera un punto original $\mathbf{p} = [x, y, 1]^T$ y un punto rotado $\mathbf{p}' = [x', y', 1]^T$ tras aplicar una rotación de ángulo θ alrededor del origen. Para esto, primero, se expresa el punto $\mathbf{p}(x, y)$ en coordenadas polares, donde r es la distancia al origen y ϕ es el ángulo que forma el punto con el eje x :

$$x = r \cos(\phi) \quad y = r \sin(\phi)$$

Tras aplicar una rotación de ángulo θ , el nuevo punto $\mathbf{p}'(x', y')$ tiene las siguientes coordenadas polares:

$$x' = r \cos(\theta + \phi) \quad y' = r \sin(\theta + \phi)$$

Usando las identidades trigonométricas de la suma de ángulos:

$$\begin{aligned}\cos(\theta + \phi) &= \cos(\theta) \cos(\phi) - \sin(\theta) \sin(\phi) \\ \sin(\theta + \phi) &= \sin(\theta) \cos(\phi) + \cos(\theta) \sin(\phi)\end{aligned}$$

se obtiene:

$$\begin{aligned}x' &= r[\cos(\theta) \cos(\phi) - \sin(\theta) \sin(\phi)] \\ y' &= r[\sin(\theta) \cos(\phi) + \cos(\theta) \sin(\phi)]\end{aligned}$$

Dado que $x = r \cos(\phi)$ y $y = r \sin(\phi)$, al sustituir en las ecuaciones anteriores se llega a las siguientes expresiones para las coordenadas rotadas:

$$\begin{aligned}x' &= x \cos(\theta) - y \sin(\theta) \\ y' &= y \cos(\theta) + x \sin(\theta)\end{aligned}$$

Las ecuaciones de rotación también pueden representarse de manera matricial utilizando coordenadas homogéneas usando el vector inicial $\mathbf{p} = [x, y, 1]^T$ y el vector rotado $\mathbf{p}' = [x', y', 1]^T$ a través de la matriz \mathbf{R} como:

$$\mathbf{p}' = \mathbf{R}\mathbf{p}$$

donde:

$$\mathbf{p}' \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Multiplicando \mathbf{p} por \mathbf{R} , se obtiene \mathbf{p}' :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Estas identidades permiten descomponer la rotación en términos de funciones trigonométricas y expresar las nuevas coordenadas en función de las originales.

La Figura 10.10 se muestra un ejemplo de rotación de una imagen. Nótese, en este caso, que las dimensiones de la imagen rotada se ha modificado.



Figura 10.10: A la izquierda: imagen original tomada en el Museo Botero de la ciudad de Bogotá de sus obras “La Carta” y “Naranjas”. A la derecha: ejemplo de imagen rotada.

10.4.6. Algoritmo Inverso de Rotación

Para girar una imagen, se sigue un proceso que implica la transformación de coordenadas y el ajuste del tamaño de misma. A continuación, se describe el algoritmo paso a paso utilizando coordenadas homogéneas.

Sea I la imagen original con dimensiones $M \times N$, donde cada píxel está en un punto de coordenadas homogéneas:

$$\mathbf{p} = [x \ y \ 1]^T$$

La matriz de rotación $R(\theta)$ y $R^{-1}(\theta)$ para un ángulo θ son respectivamente:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R^{-1}(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Esta transformación gira los puntos \mathbf{p} de la imagen original I en torno al origen $(0, 0)$. Dado que la imagen no está centrada en él, se deben aplicar transformaciones adicionales, por lo que se requiere el centro de I denominado \mathbf{p}_c :

$$\mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix} \quad \mathbf{p}_c = \frac{\mathbf{z}}{2} = \begin{bmatrix} \frac{N}{2} \\ \frac{M}{2} \\ \frac{1}{2} \end{bmatrix}$$

Para calcular el tamaño de la nueva imagen girada I' con dimensiones $M' \times N'$, donde M' es la nueva altura y N' el ancho, se utiliza la rotación de las dimensiones de la imagen original $M \times N$ mediante la matriz de rotación ya definida en valor absoluto $|R(\theta)|$. Así, las nuevas dimensiones se calculan como:

$$\mathbf{z}' = \mathbf{D}\mathbf{z}$$

donde:

$$\mathbf{z}' = \begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} |\cos(\theta)| & |\sin(\theta)| & 0 \\ |\sin(\theta)| & |\cos(\theta)| & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix}$$

Estas aseguran que la nueva imagen I' sea lo suficientemente grande para contener toda la versión girada de I . El centro de rotación \mathbf{p}'_c de I' se define como:

$$\mathbf{p}'_c = \frac{\mathbf{z}'}{2} = \begin{bmatrix} \frac{N'}{2} \\ \frac{M'}{2} \\ \frac{1}{2} \end{bmatrix}$$

Para cada píxel $\mathbf{p}' = [x' \ y' \ 1]^T$ de la imagen girada I' , se realiza el siguiente proceso:

Se calcula la posición relativa del píxel en la imagen girada I' respecto a su centro \mathbf{p}'_c :

$$\mathbf{p}'_{\text{rel}} = \mathbf{p}' - \mathbf{p}'_c$$

Aplicando la matriz de rotación inversa $R^{-1}(\theta)$ a las coordenadas relativas de la imagen girada I' :

$$\mathbf{p}_{\text{rel}} = R^{-1}(\theta) \cdot \mathbf{p}'_{\text{rel}}$$

Finalmente, determina el punto \mathbf{p} sumando la referencia \mathbf{p}_{rel} con las coordenadas del centro \mathbf{p}_c de la imagen original I :

$$\mathbf{p} = \mathbf{p}_{\text{rel}} + \mathbf{p}_c$$

Esto da las coordenadas (x, y) del píxel en la imagen original I . Es de destacar que las coordenadas obtenidas pueden no ser enteras, por lo que se redondean al más cercano:

$$x = \text{round}(x) \quad y = \text{round}(y)$$

Si las coordenadas redondeadas $I(x, y)$ están dentro de los límites de la imagen original, es decir:

$$1 \leq x \leq N \quad 1 \leq y \leq M$$

Entonces el valor de la intensidad en la posición $I(x, y)$ de la imagen original se copia a la posición $I'(x', y')$ de la imagen girada. Matemáticamente, esto se expresa como:

$$I'(y', x') = I(x, y)$$

Es importante notar que este método inverso de rotación garantiza que todos los píxeles de la imagen girada tengan un valor asignado, ya que se calcula la correspondencia para cada píxel de I' en la imagen original I lo que evita la aparición de agujeros.^{en} la imagen resultante, ya que se calcula un valor para cada píxel de la imagen girada a partir de la imagen original.

Ejemplo de Rotación

Dada una imagen I de tamaño 3×3 , se desea aplicar una rotación de 30° en sentido antihorario usando el algoritmo inverso.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Solución

La imagen original I tiene las dimensiones:

$$M = 3 \quad (\text{filas}) \quad N = 3 \quad (\text{columnas})$$

La matriz de \mathbf{D} es:

$$\mathbf{D} = \begin{bmatrix} |\cos(30^\circ)| & |-\sin(30^\circ)| & 0 \\ |\sin(30^\circ)| & |\cos(30^\circ)| & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para determinar el tamaño de la imagen rotada I' , se aplica la matriz \mathbf{D} a las dimensiones originales $M = 3$ y $N = 3$ de la imagen I la cual tendrá las nuevas dimensiones $M' \times N'$. Esto se expresa mediante la transformación:

$$\mathbf{z}' = \mathbf{D}\mathbf{z}$$

donde:

$$\mathbf{z}' = \begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} \quad \mathbf{z}' = \mathbf{D}\mathbf{z} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 4.0981 \\ 4.0981 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \\ 1 \end{bmatrix}$$

Por lo tanto, las nuevas dimensiones de la imagen son al redondear hacia arriba:

$$M' = 3 \quad (\text{filas}) \quad N' = 3 \quad (\text{columnas})$$

Redondeando, se obtiene el tamaño de la imagen rotada I' es $M' = 5$ y $N' = 5$, es decir de 5×5 .

Los centros de las imágenes original $I(x, y)$ y rotada $I'(x', y')$ son, respectivamente:

$$\mathbf{p}_c = \frac{\mathbf{z}}{2} = \begin{bmatrix} \frac{N}{2} \\ \frac{M}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{3}{2} \\ \frac{3}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 1.5 \\ 1 \end{bmatrix} \quad \mathbf{p}'_c = \begin{bmatrix} \frac{N'}{2} \\ \frac{M'}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{5}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 2.5 \\ 1 \end{bmatrix}$$

La matriz de rotación $R(\theta)$ y $R^{-1}(\theta)$ para un ángulo $\theta = 30^\circ$ son respectivamente:

$$R(\theta) = \begin{bmatrix} \cos(30^\circ) & -\sin(30^\circ) & 0 \\ \sin(30^\circ) & \cos(30^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R^{-1}(\theta) = \begin{bmatrix} \cos(30^\circ) & \sin(30^\circ) & 0 \\ -\sin(30^\circ) & \cos(30^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para cada píxel en la posición (x', y') en la imagen rotada de 5×5 , se busca la posición (x, y) que tiene la intensidad correspondiente. Para poder rotar cómodamente la imagen si desbordarse, se recurre a las coordenadas relativas haciendo una translación al centro:

$$\mathbf{p}'_{\text{rel}} = \mathbf{p}' - \mathbf{p}'_c = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} - \begin{bmatrix} 2.5 \\ 2.5 \\ 1 \end{bmatrix} = \begin{bmatrix} x' - 2.5 \\ y' - 2.5 \\ 0 \end{bmatrix}$$

Para cada píxel en la nueva imagen I' , se calcula su posición en la imagen original I mediante la matriz inversa $R^{-1}(\theta)$:

Ejemplo 1: Para las coordenadas escaladas $\mathbf{p}' = [1 \ 1 \ 1]^T$, se aplica la inversa de la matriz de rotación y se redondea $\lfloor \mathbf{p} \rfloor$:

Las coordenadas relativas se calculan como:

$$\mathbf{p}'_{\text{rel}} = \begin{bmatrix} x' - 2.5 \\ y' - 2.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 - 2.5 \\ 1 - 2.5 \\ 0 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.5 \\ 0 \end{bmatrix}$$

Ahora, se aplica la matriz de rotación inversa:

$$\mathbf{p}_{\text{rel}} = R^{-1}(\theta) \cdot \mathbf{p}'_{\text{rel}} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1.5 \\ -1.5 \\ 0 \end{bmatrix} \implies \mathbf{p}_{\text{rel}} = \begin{bmatrix} -1.799 \\ -0.699 \\ 0 \end{bmatrix}$$

Se ajusta con el centro de la imagen original:

$$\mathbf{p} = \mathbf{p}_{\text{rel}} + \mathbf{p}_c = \begin{bmatrix} -1.799 \\ -0.699 \\ 0 \end{bmatrix} + \begin{bmatrix} 1.5 \\ 1.5 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.299 \\ 0.801 \\ 1 \end{bmatrix} \implies \mathbf{p} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Dado que las coordenadas (x, y) en $I(1, 0)$ no están dentro del rango de la imagen original I , este píxel se deja en cero. Por lo tanto, la intensidad se reemplaza en $I'(1, 1) = 0$.

Ejemplo 2; Para las coordenadas escaladas $\mathbf{p}' = [3 \ 4 \ 1]^T$, se aplica la inversa de la matriz de rotación y se redondea $\lfloor \mathbf{p} \rfloor$:

Las coordenadas relativas se calculan como:

$$\mathbf{p}'_{\text{rel}} = \begin{bmatrix} x' - 2.5 \\ y' - 2.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 - 2.5 \\ 4 - 2.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.5 \\ 1 \end{bmatrix}$$

Ahora, se aplica la matriz de rotación inversa:

$$\mathbf{p}_{\text{rel}} = R^{-1}(\theta) \cdot \mathbf{p}'_{\text{rel}} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 1.5 \\ 1 \end{bmatrix} \implies \mathbf{p}_{\text{rel}} = \begin{bmatrix} 1.183 \\ 1.049 \\ 1 \end{bmatrix}$$

Se ajusta con el centro de la imagen original:

$$\mathbf{p} = \mathbf{p}_{\text{rel}} + \mathbf{p}_c = \begin{bmatrix} 1.183 \\ 1.049 \\ 1 \end{bmatrix} + \begin{bmatrix} 1.5 \\ 1.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.68 \\ 2.55 \\ 2 \end{bmatrix} \implies \mathbf{p} = \begin{bmatrix} 3 \\ 3 \\ 2 \end{bmatrix}$$

La intensidad en $I(3, 3) = 9$ se reemplaza en $I'(4, 3) = 5$.

La Tabla 10.3 resume los resultados de todos los valores mapeados matricialmente para el algoritmo inverso de rotación donde se presenta la posición en la matriz transladada (y', x') , la encontrada en la matriz fuente (y, x) , la redondeada $\lfloor (y, x) \rfloor$ y finalmente la intensidad correspondiente $I'(y', x') = I(y, x)$.

La matriz de la imagen rotada I' después de aplicar el algoritmo es:

$$I' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 4 & 5 & 3 & 0 \\ 0 & 8 & 9 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 \end{bmatrix}$$

(y', x')	(y, x)	$\lfloor (y, x) \rfloor$	$I'(y', x)$	(y', x')	(x, x)	$\lfloor (x, x) \rfloor$	$I'(x', x')$
(1, 1)	(0.95, -0.55)	(1, -1)	0	(3, 1)	(2.68, 0.45)	(3, 0)	0
(1, 2)	(0.45, 0.32)	(0, 0)	0	(3, 2)	(2.18, 1.32)	(2, 1)	4
(1, 3)	(-0.05, 1.18)	(0, 1)	0	(3, 3)	(1.68, 2.18)	(2, 2)	5
(1, 4)	(-0.55, 2.05)	(-1, 2)	0	(3, 4)	(1.18, 3.05)	(1, 3)	3
(1, 5)	(-1.05, 2.92)	(-1, 3)	0	(3, 5)	(0.68, 3.92)	(1, 4)	0
(2, 1)	(1.82, -0.05)	(2, 0)	0	(4, 1)	(3.55, 0.95)	(4, 1)	0
(2, 2)	(1.32, 0.82)	(1, 1)	1	(4, 2)	(3.05, 1.82)	(3, 2)	8
(2, 3)	(0.82, 1.68)	(1, 2)	2	(4, 3)	(2.55, 2.68)	(3, 3)	9
(2, 4)	(0.32, 2.55)	(0, 3)	0	(4, 4)	(2.05, 3.55)	(2, 4)	0
(2, 5)	(-0.18, 3.42)	(0, 3)	0	(4, 5)	(1.55, 4.42)	(2, 4)	0
(3, 1)	(2.68, 0.45)	(3, 0)	0	(5, 1)	(4.42, 1.45)	(4, 1)	0
(3, 2)	(2.18, 1.32)	(2, 1)	4	(5, 2)	(3.92, 2.32)	(4, 2)	0
(3, 3)	(1.68, 2.18)	(2, 2)	5	(5, 3)	(3.42, 3.18)	(3, 3)	9
(3, 4)	(1.18, 3.05)	(1, 3)	3	(5, 4)	(2.92, 4.05)	(3, 4)	0
(3, 5)	(0.68, 3.92)	(1, 4)	0	(5, 5)	(2.42, 4.92)	(2, 5)	0

Cuadro 10.3: Tabla de resultados de la rotación. Se presenta la posición recorrida en el destino (x', y') , la encontrada en la fuente (x, y) por el algoritmo, la redondeada $\lfloor (x, y) \rfloor$ y finalmente la intensidad correspondiente $I'(x', y')=I(x, y)$.

10.4.7. Ejemplo en Python: Transformaciones afines básicas

El siguiente código realiza transformaciones básicas en una imagen utilizando Python. Primero, carga una imagen en formato RGB y reduce su resolución aplicando un factor de escala. Luego, aplica cuatro operaciones fundamentales de procesamiento de imágenes: rotación en 60 grados, escalado a un tamaño menor, recorte de una región específica y traslación de la imagen recortada a una nueva posición. Finalmente, muestra los resultados de cada transformación en una visualización comparativa.

Solución

```

1
2  """
3  #Importación de Librería Básicas
4  import matplotlib.pyplot as plt
5  import numpy as np
6  import sys
7
8  from ip_functions import *
9
10 # Open image
11 RGB=plt.imread("cartagena.jpg")
12 RGB=np.array(RGB);
13 # Factor de Escala S
14 S=5
15 RGB=RGB[0::S,0::S,:]
16 #Convertir a escala de grises
17 #RGB=rgb2gray(RGB)
18
19 Rotado=imrotate(RGB, 60)
20 plt.figure(1)
21 plt.subplot (2,2,1)
22 plt.title("Rotado")
23 plt.imshow(Rotado,cmap='gray')
24
25 Escalado=imresize(RGB, 0.1)
26 plt.subplot (2,2,2)
27 plt.title("Escalado")
28 plt.imshow(Escalado,cmap='gray')
29
30 Cortar=imcrop(RGB,400,65,250,250)

```

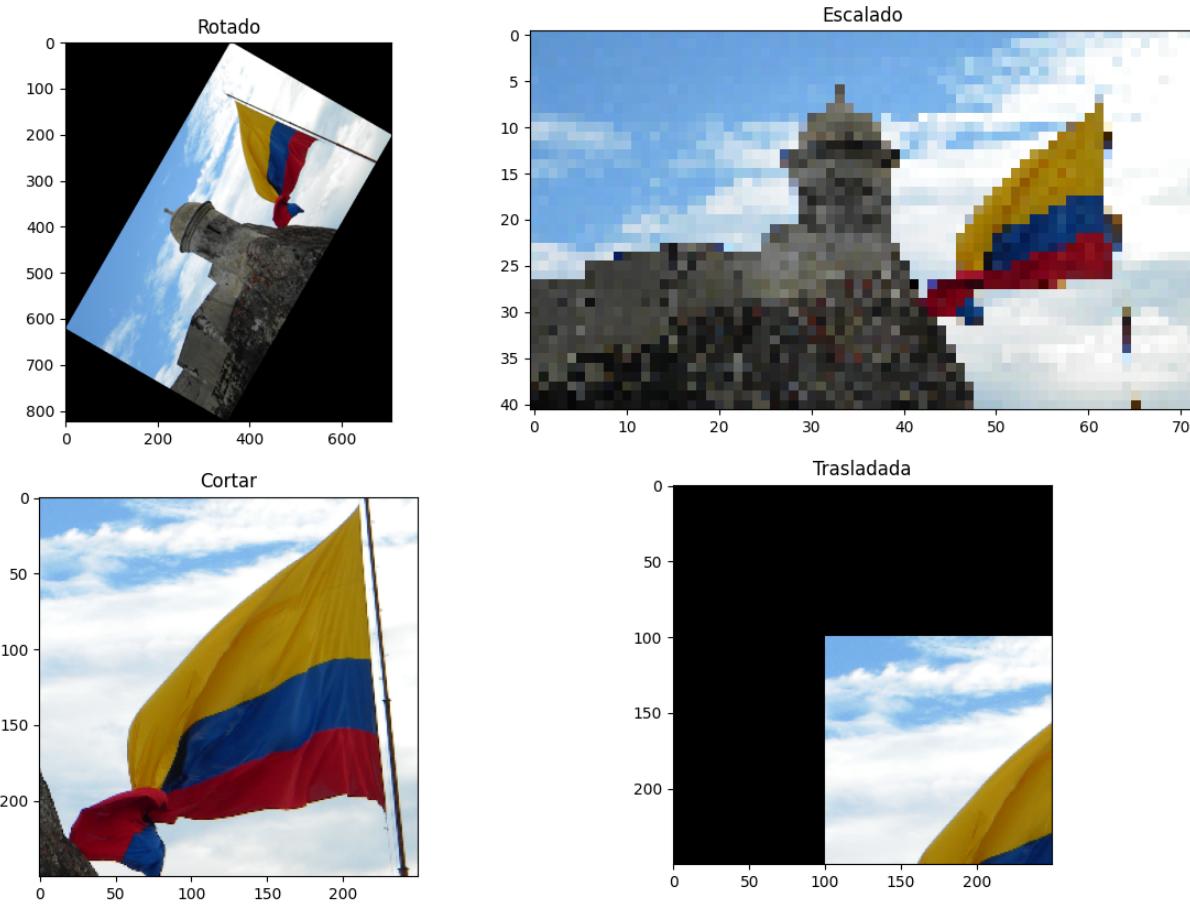


Figura 10.11: Transformaciones aplicadas a una imagen original. (a) Rotación de 60 grados, donde la imagen se gira manteniendo su estructura. (b) Escalado, reduciendo el tamaño de la imagen mediante un factor determinado. (c) Recorte de una región específica de la imagen original, seleccionando un área de interés. (d) Traslación de la imagen recortada a una nueva posición dentro del mismo lienzo.

```

31 plt.subplot (2,2,3)
32 plt.title("Cortar")
33 plt.imshow(Cortar,cmap='gray')
34
35 Trasladada=imtranslate(Cortar,(100,100),'same')
36 plt.subplot (2,2,4)
37 plt.title("Trasladada")
38 plt.imshow(Trasladada,cmap='gray')
39
40 plt.show()

```

La Fig.10.11 muestra el resultado de diversas transformaciones aplicadas a una imagen original. En la primera parte, se presenta la imagen rotada 60 grados. La segunda subimagen ilustra el resultado del escalado, reduciendo su tamaño. En la tercera, se observa un recorte de una región específica de la imagen, y finalmente, en la cuarta región, se aprecia la traslación de la imagen recortada a una nueva ubicación. Estas operaciones permiten modificar la imagen para distintos análisis y aplicaciones en procesamiento digital de imágenes. Las funciones referidas allí, se encuentran en el capítulo 13.

10.4.8. Cizallamiento

En el proceso de cizallamiento geométrico, cada píxel de la imagen fuente con coordenadas $\mathbf{p} = [x, y, 1]^T$ se reasigna a una nueva posición en la imagen destino, $\mathbf{p}' = [x', y', 1]^T$, desplazando cada punto en una dirección fija, en una

cantidad proporcional a su distancia desde una línea de referencia. En la Fig.10.12 se ilustra el cizallamiento de una forma (línea continua) a una posición final (línea discontinua) dentro de una escena [13,15].

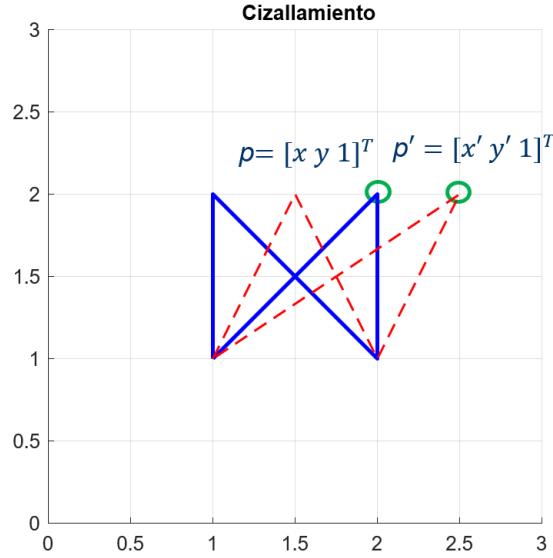


Figura 10.12: Representación del Cizallamiento de una forma (línea continua) a una forma final (línea discontinua) dentro de una escena. Nótese que, aunque las líneas cambian de ángulo, se mantiene el paralelismo final.

Este caso puede expresarse mediante las ecuaciones:

$$x' = x + C_x \cdot y \quad y' = y + C_y \cdot x$$

donde C_x y C_y son los factores de cizallamiento en las direcciones x y y , respectivamente. Para representarlo de manera matricial, se utiliza la notación homogénea, en la que tanto \mathbf{p} como \mathbf{p}' son vectores y \mathbf{C} es la matriz de cizallamiento. Así, la transformación puede escribirse como:

$$\mathbf{p}' = \mathbf{C}\mathbf{p}$$

donde:

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & C_y & 0 \\ C_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplicando \mathbf{C} por \mathbf{p} , se obtiene:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & C_y & 0 \\ C_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Este producto matricial genera las nuevas coordenadas directas cizalladas de la imagen transformada. En muchos casos, se aplica solo horizontal ($C_y = 0$) o vertical ($C_x = 0$), para esto las matrices horizontal \mathbf{C}_H y vertical \mathbf{C}_V serían, respectivamente:

$$\mathbf{C}_H = \begin{bmatrix} 1 & 0 & 0 \\ C_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{C}_V = \begin{bmatrix} 1 & C_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La transformación puede aplicarse en ambas direcciones simultáneamente, lo que resulta en una transformación más compleja que combina los efectos. La Figura 10.13 se muestra un ejemplo de sizamiento de una imagen. Nótese, en este caso, que las dimensiones de la imagen Cizallada se ha modificado.



Figura 10.13: A la izquierda: imagen original tomada en el Museo Botero de la ciudad de Bogotá de sus obras “La Carta” y “Naranjas”. A la derecha: ejemplo de imagen Cizallada.

10.4.9. Algoritmo Inverso de cizallamiento

El proceso de cizallamiento de una imagen implica deformar la imagen original mediante factores horizontales y verticales. Sea I la imagen original con dimensiones $N \times M$, donde cada píxel de la imagen original $I(x, y)$ está en un punto de coordenadas homogéneas:

$$\mathbf{p} = [x \ y \ 1]^T$$

Para calcular el tamaño de la nueva imagen girada I' con dimensiones $M' \times N'$, donde M' es la nueva altura y N' el ancho, se utiliza la rotación de las dimensiones de la imagen original $M \times N$ mediante la matriz de Cizalla \mathbf{C} . Así, las nuevas dimensiones se calculan como:

$$\mathbf{z}' = \mathbf{C}\mathbf{z}$$

donde:

$$\mathbf{z}' = \begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & C_y & 0 \\ C_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & C_y & 0 \\ C_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} N \\ M \\ 1 \end{bmatrix}$$

Para cada píxel en la nueva imagen I' , se calcula la posición correspondiente en la imagen original I . Así, las coordenadas homogéneas \mathbf{p}' de un píxel en la imagen cizallada se expresan como:

$$\mathbf{p}' = [x' \quad y' \quad 1]^T$$

Las coordenadas originales \mathbf{p} en la imagen I se obtienen aplicando la inversa de la matriz de cizallamiento \mathbf{C} :

$$\mathbf{p} = \mathbf{C}^{-1}\mathbf{p}' = \frac{1}{1 - C_x C_y} \begin{bmatrix} 1 & -C_y & 0 \\ -C_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Donde la inversa de la matriz de cizallamiento \mathbf{C}^{-1} es:

$$\mathbf{C}^{-1} = \frac{1}{1 - C_x C_y} \begin{bmatrix} 1 & -C_y & 0 \\ -C_x & 1 & 0 \\ 0 & 0 & 1 - C_x C_y \end{bmatrix}$$

Las coordenadas originales (x, y) se redondean al píxel más cercano:

$$x = \text{round}(x) \quad y = \text{round}(y)$$

Finalmente, se verifica si estas están dentro de los límites de la imagen original I :

$$1 \leq x \leq M \quad 1 \leq y \leq N$$

Si las coordenadas (x, y) están en los límites, se asigna el valor del píxel de la imagen original $I(x, y)$ al píxel correspondiente en la imagen cizallada $I'(x', y')$, lo cual se expresa como:

$$I'(y', x') = I(x, y)$$

Este método de inverso asegura que cada píxel en la imagen cizallada tenga un valor asignado, manteniendo la integridad de la imagen original en la región donde se superponen.

1.3.8.1 Ejemplo Cizallamiento

Dada una imagen I de tamaño 3×3 , se desea aplicar un cizallamiento horizontal con un factor $C_x = 0.5$ usando el algoritmo inverso.

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Solución

Sea I la imagen original con dimensiones $N \times M$, donde cada píxel de la imagen original $I(x, y)$ está en un punto de coordenadas homogéneas:

$$\mathbf{p} = [x \quad y \quad 1]^T$$

Para realizar un cizallamiento horizontal con un factor $C_x = 0.5$, se emplea la siguiente matriz de cizalla \mathbf{C} :

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para calcular las nuevas dimensiones de I' , se aplica la matriz de cizalla a las dimensiones originales, se substituyen las dimensiones originales $M = 3$ y $N = 3$:

donde:

$$\mathbf{z}' = \begin{bmatrix} N' \\ M' \\ 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} N \\ M \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} \quad \mathbf{z}' = \mathbf{C}\mathbf{z} = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 4.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 1 \end{bmatrix}$$

Por lo tanto, las nuevas dimensiones de la imagen son al redondear hacia arriba:

$$M' = 3 \quad (\text{filas}) \quad N' = 5 \quad (\text{columnas})$$

Para cada píxel en la nueva imagen I' , se calcula su posición en la imagen original I . Las coordenadas originales \mathbf{p} se obtienen aplicando la inversa de la matriz de cizallamiento \mathbf{C}^{-1} :

$$\mathbf{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ejemplo 1: Para las coordenadas cizalladas $\mathbf{p}' = [3 \ 2 \ 1]^T$, se aplica la inversa de la matriz de cizallamiento y se redondea $\lfloor \mathbf{p} \rfloor$:

$$\mathbf{p} = \mathbf{C}^{-1}\mathbf{p}' = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \implies \mathbf{p} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

La intensidad en $I(2, 2) = 5$ se reemplaza en $I'(2, 3) = 5$.

Ejemplo 2: Para las coordenadas cizalladas $\mathbf{p}' = [1 \ 3 \ 1]^T$, se aplica la inversa de la matriz de cizallamiento y se redondea $\lfloor \mathbf{p} \rfloor$:

$$\mathbf{p} = \mathbf{C}^{-1}\mathbf{p}' = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2.5 \\ 1 \end{bmatrix} \implies \mathbf{p} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

La intensidad en $I(3, 1) = 3$ se reemplaza en $I'(3, 1) = 3$.

(y', x')	(y, x)	(y, x) redondeado	$I_{\text{sheared}}(y', x')$	(y', x')	(y, x)	(y, x) redondeado
(1, 1)	(1, 0.5)	(1, 1)	1	(2, 4)	(2, 3)	(2, 3)
(1, 2)	(1, 1.5)	(1, 2)	2	(2, 5)	(2, 4)	(2, 4)
(1, 3)	(1, 2.5)	(1, 3)	3	(3, 1)	(3, -0.5)	(3, -1)
(1, 4)	(1, 3.5)	(1, 4)	0	(3, 2)	(3, 0.5)	(3, 1)
(1, 5)	(1, 4.5)	(1, 5)	0	(3, 3)	(3, 1.5)	(3, 2)
(2, 1)	(2, 0)	(2, 0)	0	(3, 4)	(3, 2.5)	(3, 3)
(2, 2)	(2, 1)	(2, 1)	4	(3, 5)	(3, 3.5)	(3, 4)
(2, 3)	(2, 2)	(2, 2)	5			

Cuadro 10.4: Tabla de resultados de la cizallamiento. Se resumen los resultados de todos los valores mapeados matricialmente para el algoritmo inverso de cizallamiento donde se presenta la posición en la matriz transladada (y', x') , la encontrada en la matriz fuente (y, x) , la redondeada $\lfloor(y, x)\rfloor$ y finalmente la intensidad correspondiente $I'(y', x')=I(y, x)$.

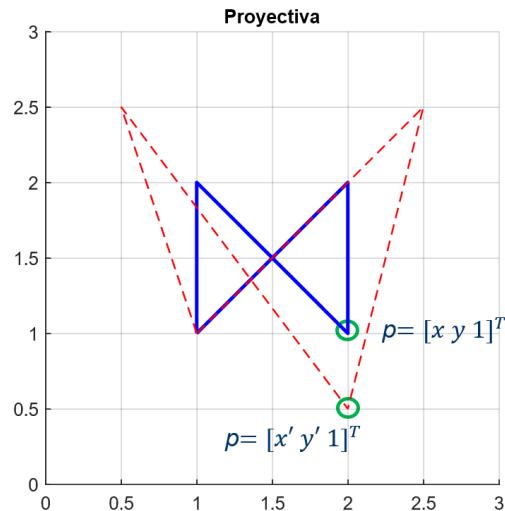
La Tabla 10.4 Se resumen los resultados de todos los valores mapeados matricialmente para el algoritmo inverso de cizallamiento donde se presenta la posición en la matriz transladada (y', x') , la encontrada en la matriz fuente (y, x) , la redondeada $\lfloor(y, x)\rfloor$ y finalmente la intensidad correspondiente $I'(y', x')=I(y, x)$

La matriz de la imagen cizallada I' después de aplicar el algoritmo es:

$$I' = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \end{bmatrix}$$

10.5. Transformación Proyectiva

Es un tipo de transformación geométrica que mapea puntos de un plano a otro, preservando las líneas rectas de la imagen original después de la transformación. Sin embargo, esta transformación no preserva las longitudes ni los ángulos, por lo que pueden transformar líneas paralelas en líneas que se cruzan en un punto del infinito, lo que genera un cambio de perspectiva. Esta es más general que las afines y puede representar también efectos como rotación, escalado, traslación, y distorsiones más complejas como la perspectiva [13, 15].



A la derecha:

Figura 10.14: Representación de la proyección de una forma (línea continua) a una posición final (línea discontinua) dentro de una escena. Nótese que, algunas líneas cambiar de longitud y otras pierden el paralelismo final.

Para realizar la transformación proyectiva, se seleccionan cuatro puntos en la imagen original I , denotados como $\mathbf{p}_i = (x_i, y_i)$, con $i = 1, 2, 3, 4$, que son los clave que se quieren mover. Luego, se seleccionan otros cuatro puntos correspondientes en la imagen transformada I' , marcados como $\mathbf{p}'_i = (x'_i, y'_i)$ para un total de ocho. El objetivo de la transformación proyectiva es generar una mapeo de los puntos seleccionados en I hacia los puntos correspondientes en I' .

Así, los puntos en la imagen original son:

$$\mathbf{p}_1 = (x_1, y_1) \quad \mathbf{p}_2 = (x_2, y_2) \quad \mathbf{p}_3 = (x_3, y_3) \quad \mathbf{p}_4 = (x_4, y_4)$$

Los puntos en la imagen transformada son:

$$\mathbf{p}'_1 = (x'_1, y'_1) \quad \mathbf{p}'_2 = (x'_2, y'_2) \quad \mathbf{p}'_3 = (x'_3, y'_3) \quad \mathbf{p}'_4 = (x'_4, y'_4)$$

La transformación proyectiva se define mediante una matriz homográfica \mathbf{H} de 3x3, que mapea puntos del plano $\mathbf{p} = [x, y, 1]^T$ a puntos transformados $\mathbf{p}' = [x', y', w]^T$:

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \quad \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (10.2)$$

Las coordenadas transformadas se obtienen normalizando por w :

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \quad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \quad (10.3)$$

Donde $w = h_{31}x + h_{32}y + h_{33}$ actúa como factor de normalización.

Aunque \mathbf{H} tiene 9 elementos, es homogénea, lo que permite fijar $h_{33} = 1$, resultando en solo 8 grados de libertad. Consecuentemente, cada par de puntos correspondientes \mathbf{p}_i y \mathbf{p}'_i genera dos ecuaciones:

$$\mathbf{p}'_i = \mathbf{H}\mathbf{p}_i \quad \begin{bmatrix} x'_i \\ y'_i \\ w_i \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \quad (10.4)$$

Desarrollando estas ecuaciones para ocho puntos (cuatro pares de puntos) (x_i, y_i) y (x'_i, y'_i) , $i = 1, 2, 3, 4$, se obtiene:

$$x'_i(h_{31}x_i + h_{32}y_i + 1) = h_{11}x_i + h_{12}y_i + h_{13} \quad (10.5)$$

$$y'_i(h_{31}x_i + h_{32}y_i + 1) = h_{21}x_i + h_{22}y_i + h_{23} \quad (10.6)$$

El sistema completo de ocho ecuaciones puede organizarse en forma matricial para resolver los valores de $h_{11}, h_{12}, \dots, h_{33}$. Dado que $h_{33} = 1$ para garantizar que la matriz sea homogénea:

$$x'_i(h_{31}x_i + h_{32}y_i + 1) - h_{11}x_i - h_{12}y_i - h_{13} = 0$$

$$y'_i(h_{31}x_i + h_{32}y_i + 1) - h_{21}x_i - h_{22}y_i - h_{23} = 0$$

Para los cuatro pares de puntos $\mathbf{p} = [x, y, 1]^T$ y $\mathbf{p}' = [x', y', w]^T$, se obtienen:

$$\begin{aligned}x'_1(h_{31}x_1 + h_{32}y_1 + 1) - h_{11}x_1 - h_{12}y_1 - h_{13} &= 0 \\y'_1(h_{31}x_1 + h_{32}y_1 + 1) - h_{21}x_1 - h_{22}y_1 - h_{23} &= 0\end{aligned}$$

$$\begin{aligned}x'_2(h_{31}x_2 + h_{32}y_2 + 1) - h_{11}x_2 - h_{12}y_2 - h_{13} &= 0 \\y'_2(h_{31}x_2 + h_{32}y_2 + 1) - h_{21}x_2 - h_{22}y_2 - h_{23} &= 0\end{aligned}$$

$$\begin{aligned}x'_3(h_{31}x_3 + h_{32}y_3 + 1) - h_{11}x_3 - h_{12}y_3 - h_{13} &= 0 \\y'_3(h_{31}x_3 + h_{32}y_3 + 1) - h_{21}x_3 - h_{22}y_3 - h_{23} &= 0\end{aligned}$$

$$\begin{aligned}x'_4(h_{31}x_4 + h_{32}y_4 + 1) - h_{11}x_4 - h_{12}y_4 - h_{13} &= 0 \\y'_4(h_{31}x_4 + h_{32}y_4 + 1) - h_{21}x_4 - h_{22}y_4 - h_{23} &= 0\end{aligned}$$

Estas se pueden organizar en forma matricial como se ve en (10.7):

$$\left[\begin{array}{ccccccc} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x'_3x_3 & -x'_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y'_3x_3 & -y'_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4x_4 & -y'_4y_4 \end{array} \right] \left[\begin{array}{c} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{array} \right] = \left[\begin{array}{c} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{array} \right] \quad (10.7)$$

donde \mathbf{A} es la matriz 8×8 de coeficientes, \mathbf{H} es el vector de 8×1 de incógnitas (los elementos de la matriz de transformación proyectiva), y \mathbf{b} es el vector 8×1 de términos independientes. Para resolver este sistema, se pueden utilizar métodos numéricos o la eliminación gaussiana, entre otros.

$$\mathbf{A} \cdot \mathbf{H} = \mathbf{b}$$

Así, finalmente, se llega a la matriz proyectiva:

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$

10.5.1. Ejemplo en Python: Transformaciones proyectivas

Este código implementa una transformación proyectiva aplicada a una imagen de entrada, en este caso, una obra de Botero. El proceso inicia con la selección de un conjunto de puntos de referencia en la imagen original y sus correspondientes ubicaciones en la imagen transformada. A partir de estos puntos, se calcula la matriz de transformación proyectiva H , la cual permite mapear la imagen de entrada a su versión transformada. Para mejorar la eficiencia computacional, se aplica un factor de escala que reduce la resolución de la imagen antes de la transformación. Finalmente, el código emplea interpolaciones para reconstruir la imagen transformada y facilitar su visualización.

Solución

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import sys
4
5
6 from ip_functions_old import *
7
8 # Leer la imagen a proyectar
9 I= plt.imread('boterol.jpg') # ruta de tu imagen
10
11 # Factor de Escala S para reducir carga computacional
12 S=2
13 I=I[0::S,0::S,:]
14
15 #En escala de grises
16 #I=rgb2gray(I)
17
18 #Arreglo de puntos Iniciales y Finales
19
20 # Crear el arreglo de puntos Iniciales predefinidos
21
22 puntos_iniciales = np.array([
23     [126, 20],
24     [126, 317],
25     [393, 268],
26     [394, 65]
27 ])
28
29 # Crear el arreglo de puntos finales predefinidos
30 puntos_finales = np.array([
31     [126, 20],
32     [126, 317],
33     [508, 321],
34     [508, 21]
35 ])
36
37 #Cálculo de la Matriz Proyectiva H
38
39 # Calcular la matriz de Transformación Proyectiva
40 H = fitgeotrans(puntos_iniciales, puntos_finales, transformation_type='projective')
41
42 print("Matriz de transformación H:")
43 np.set_printoptions(formatter={'float': '{:.2f}'.format})
44 print(H)
45
46 #Aplicación de la Transformación Proyectiva
47
48 # Aplicar la transformación
49 imagen_transformada = imwarp(I, H)
50
51 # Visualización antes y después
52 plt.figure(figsize=(10, 5))
53
54 # Mostrar la imagen original
55 plt.subplot(1, 2, 1)
56 plt.imshow(I,cmap='gray')
57 plt.title('Imagen Original')
58 plt.axis('off')
59
60 # Mostrar la imagen transformada
61 plt.subplot(1, 2, 2)
62 plt.imshow(imagen_transformada,cmap='gray')
63 plt.title('Imagen Proyectada')
64 plt.axis('off')
65
66 # Mostrar ambas imágenes
67 plt.tight_layout()
68 plt.show()
```

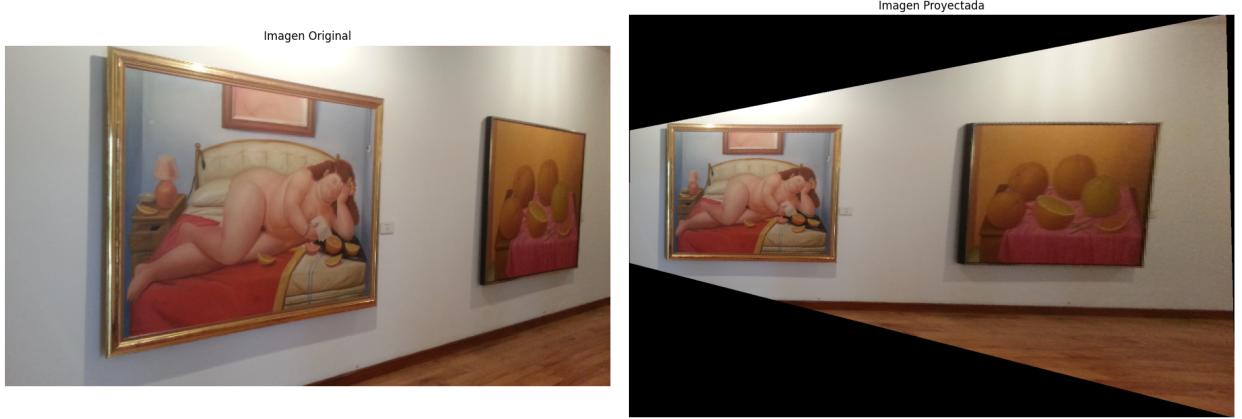


Figura 10.15: En la (a), se presenta la imagen de entrada, donde se han definido un conjunto de puntos de referencia iniciales. En (b), se observa la imagen transformada, obtenida mediante la matriz proyectiva H . La imagen original fue tomada en el Museo Botero de la ciudad de Bogotá de sus obras “La Carta” y “Naranjas” .

La Fig. 10.15 ilustra el resultado de la aplicación de una transformación proyectiva sobre una imagen original. En la primera parte, se presenta la imagen de entrada, donde se han definido un conjunto de puntos de referencia iniciales. En la segunda subimagen, se observa la imagen transformada, obtenida mediante la matriz proyectiva H , la cual ajusta la perspectiva de la imagen según los puntos de destino especificados. Esta transformación permite corregir distorsiones geométricas, modificar la perspectiva y realizar alineaciones de imágenes en diversas aplicaciones del procesamiento digital de imágenes. Las funciones empleadas en este proceso se encuentran detalladas en el capítulo 13.

10.6. Interpolaciones

10.6.1. Vecino más cercano

La técnica de vecino más cercano es la técnica más intuitiva y fácil, que consiste en asociar al punto resultante, las coordenadas del punto de cuadrícula más cercano. Esto se logra redondeando las coordenadas resultantes del píxel para que coincidan con una posición existente. Así se le asignará al píxel original las coordenadas del píxel más cercano en la cuadrícula de la nueva imagen. Este es el método de interpolación más rápido, ya que implica poco cálculo. Esto da como resultado una imagen pixelada o en bloques.

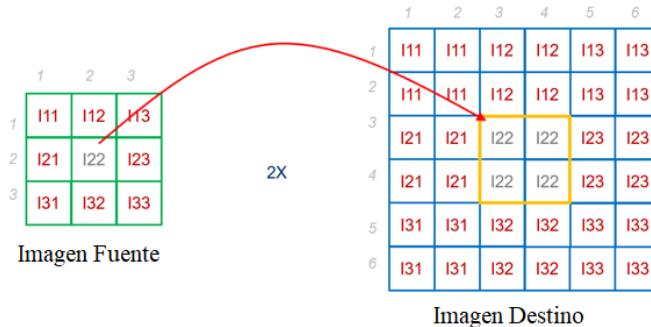


Figura 10.16: Se muestra de manera conceptual el proceso de interpolación por el Vecino más cercano. Nótese que los valores se repiten y pueden dar bordes dentados.

10.6.2. Interpolación bilineal

La interpolación bilineal determina el valor del nivel de gris a partir del promedio ponderado de los cuatro píxeles más cercanos a las coordenadas de entrada especificadas, y asigna ese valor a la coordenada de salida. Primero, se realizan dos interpolaciones lineales en una dirección y luego se calcula una interpolación lineal más en la dirección perpendicular de las otras dos rectas formada por los puntos horizontales. Puede también realizarse de manera contraria y no se altera el resultado. Si se realiza el cambio de tamaño o rotación de una imagen los resultados pueden variar significativamente según el algoritmo de interpolación que se use. No obstante, una imagen siempre pierde algo de calidad cada vez que se realice la interpolación. La Fig.10.17 muestra como el algoritmo se hacecerá al valor que se desea interpolar usando primero algunas interpolaciones lineales simples.

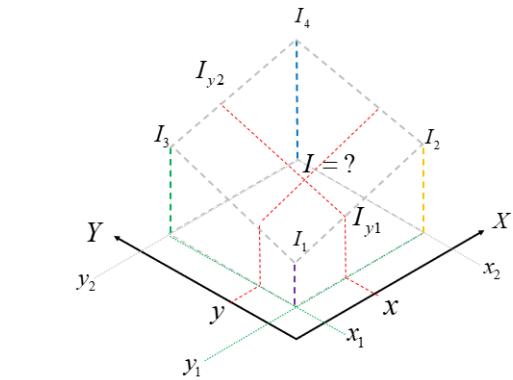


Figura 10.17: Interpolación bilineal. Estar se acerca al centro a tra'ves de inicialmente hacer interpolaciones lineales simples.

Para realizar la interpolación, como se muestra en Fig.10.17, se debe conocer la posición y la intensidad en los punto $I_1(x_1, y_1), I_2(x_2, y_2), I_3(x_1, y_2)$, y $I_4(x_2, y_1)$ vecinos de interés al punto $I(x, y)$. Inicialmente, se realiza la interpolación entre los puntos horizontales con intensidades $I_1(x_1, y_1)$ y $I_2(x_2, y_2)$, así como se muestra en la Fig.10.18:

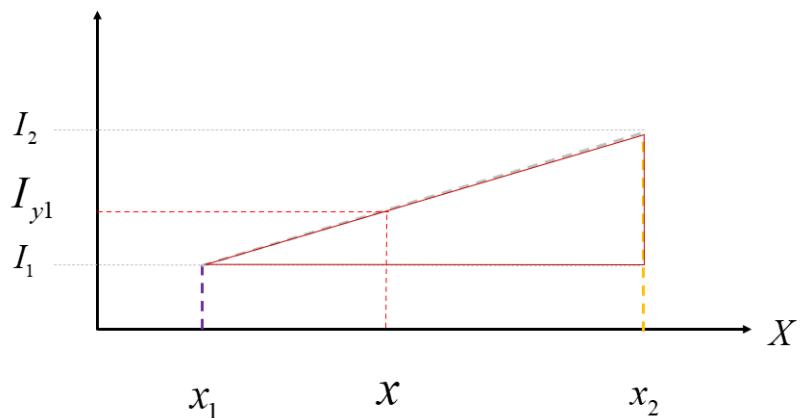


Figura 10.18: Primera aproximación lineal para I_{y1} .

Para ello se determinan las relaciones lineales y se expresa la recta I_{y1} ilustrada en (10.8):

$$\frac{I_2 - I_1}{x_2 - x_1} = \frac{I_{y_1} - I_1}{x - x_1}$$

Se puede despejar I_{y_1} :

$$I_{y_1} = I_1 + \frac{I_2 - I_1}{x_2 - x_1}(x - x_1)$$

Sumando I_1 a ambos lados:

$$I_{y_1} = I_1 + \frac{I_2 - I_1}{x_2 - x_1}(x - x_1) \quad (10.8)$$

A continuación, se repite el procedimiento para los puntos conocidos $I_3(x_1, y_2)$ y $I_4(x_2, y_2)$.

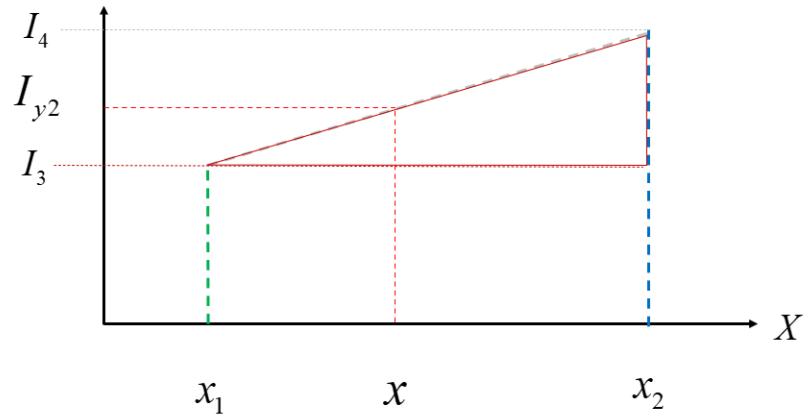


Figura 10.19: Segunda aproximación lineal para I_{y_2} .

De forma análoga se genera la relación mostrada en la Fig.10.19 y la expresión de la recta I_{y_2} mostrada en (10.9):

$$\frac{I_4 - I_3}{x_2 - x_1} = \frac{I_{y_2} - I_3}{x - x_1}$$

Se puede despejar I_{y_2} :

$$I_{y_2} = I_3 + \frac{I_4 - I_3}{x_2 - x_1}(x - x_1)$$

Sumando I_3 a ambos lados:

$$I_{y_2} = I_3 + \frac{I_4 - I_3}{x_2 - x_1}(x - x_1) \quad (10.9)$$

Finalmente, se realiza la interpolación entre los puntos ya conocidos $I_{y_1}(x, y)$ y $I_{y_2}(x, y)$ para encontrar $I(x, y)$. Véase la Fig10.20.

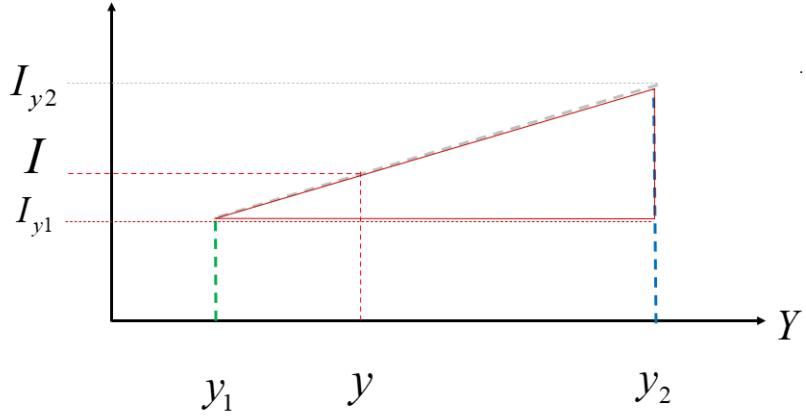


Figura 10.20: Interpolación lineal de los casos anteriores, conducente a la interpolación bilineal para I .

Así se obtiene la expresión (10.10) a partir de la Fig.10.20:

$$\frac{I_{y_2} - I_{y_1}}{y_2 - y_1} = \frac{I - I_{y_1}}{y - y_1} \quad (10.10)$$

Evaluado I_{y_1} e I_{y_2} en se encuentra la expresión general presente en (10.12) que permite la interpolación bilineal de los puntos cuatro puntos conocidos para encontrar $I(x, y)$:

$$I(x, y) = I_{y_1} + \frac{I_{y_2} - I_{y_1}}{y_2 - y_1} (y - y_1) \quad (10.11)$$

Sustituyendo las expresiones 10.13 y 10.15 en 10.11:

$$I(x, y) = \frac{(y_2 - y)}{(y_2 - y_1)} \left[\frac{(x_2 - x)I_1 + (x - x_1)I_2}{x_2 - x_1} \right] + \frac{(y - y_1)}{(y_2 - y_1)} \left[\frac{(x_2 - x)I_3 + (x - x_1)I_4}{x_2 - x_1} \right] \quad (10.12)$$

Esta ecuación realiza la interpolación bilineal en una sola expresión, utilizando los cuatro puntos conocidos I_1, I_2, I_3 , y I_4 .

Ejercicio en clase Interpolación Bilineal:

Usando interpolación bilineal determine la intensidad en $I(12, 2)$ si se definen inicialmente los puntos $I_1(11, 1) = 10$, $I_2(11, 3) = 15$, $I_3(13, 2) = 2$ y $I_4(13, 3) = 6$ como se ve en la Fig.10.20.

Solución:

Así evaluando respectivamente $I_1(11, 1) = 10$ e $I_2(11, 3) = 15$ en (10.13) y (10.14), respectivamente, se obtiene $I_{y_1}(11, 2)$:

$$m_{y_1} = \frac{I_2 - I_1}{x_2 - x_1} = \frac{15 - 10}{3 - 1} = \frac{5}{2} \quad (10.13)$$

$$I_{y_1} = \frac{5}{2}(2 - 1) + 10 = \frac{25}{2} \quad (10.14)$$

Repetiendo el proceso para $I_3(13, 2) = 2$ e $I_4(13, 3) = 6$ en (10.15) y (10.16), respectivamente, se obtiene $I_{y_2}(13, 2)$:

$$m_{y_2} = \frac{I_4 - I_3}{x_2 - x_1} = \frac{6 - 2}{3 - 1} = 2 \quad (10.15)$$

$$I_{y_2} = 2(2 - 1) + 2 = 4 \quad (10.16)$$

Finalmente, evaluando $I_{y_1}(11, 2) = 12.5$ y $I_{y_2}(13, 2) = 4$ en (10.17), se determina la intensidad correspondiente al punto $I(12, 2) = 8.25$:

$$m = \frac{4 - \frac{25}{2}}{13 - 11} = -\frac{17}{4} \approx -4.25$$

$$I = -\frac{17}{4}(12 - 11) + \frac{25}{2} = \frac{33}{4} \approx 8.25 \quad (10.17)$$

Los valores se redondean y se obtiene la solución mostrada en la Fig.10.21:

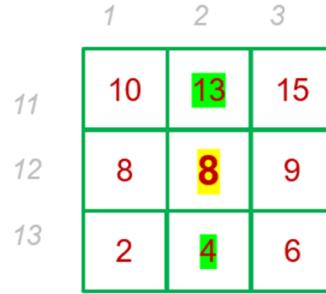


Figura 10.21: Intensidad del punto $I(12, 2)$.

10.7. Funciones

Funciones Relacionadas

Las siguientes funciones permiten modificar la geometría de una imagen mediante recorte, cambio de tamaño, rotación, traslación o transformaciones generales basadas en matrices. Estas operaciones son esenciales en procesos de preprocesamiento, registro de imágenes o análisis espacial.

1. `imcrop(I, rect)`: recorta una imagen `I` seleccionando una región rectangular definida por el parámetro `rect`, que especifica las coordenadas y dimensiones del recorte en el formato `[x, y, ancho, alto]`. Esta función se utiliza para extraer una región de interés sin modificar su escala o forma.
2. `imresize(I, scale_or_size)`: cambia el tamaño de una imagen `I`, ya sea especificando un factor de escala (por ejemplo, 0.5 para reducir a la mitad) o una nueva dimensión deseada (como [256, 256]). También puede incluir un parámetro opcional para el método de interpolación, como `'bilinear'`, `'nearest'` o `'bicubic'`.
3. `imrotate(I, angle)`: rota una imagen `I` un ángulo en grados definido por el parámetro `angle`. El sentido de rotación es antihorario por defecto. El parámetro `method` define el tipo de interpolación, y `crop=True` permite recortar la imagen resultante al tamaño original.
4. `imtranslate(I, translation)`: desplaza una imagen `I` mediante el vector de traslación `translation = [dx, dy]`, moviendo su contenido horizontal y verticalmente sin cambiar la forma de la imagen. Se puede definir el tipo de interpolación y el tratamiento de los bordes mediante parámetros adicionales.
5. `fitgeotrans(movingPoints, fixedPoints, tipo)`: estima una transformación geométrica entre dos conjuntos de puntos de control, `movingPoints` y `fixedPoints`, devolviendo un objeto que contiene la matriz de transformación `H`. El parámetro `tipo` define el tipo de transformación (por ejemplo, `'affine'`, `'projective'` o `'similarity'`).
6. `imwarp(I, tform)`: aplica una transformación geométrica a la imagen `I` utilizando un objeto de transformación `tform`, generalmente generado por `fitgeotrans`. La función realiza la interpolación de los nuevos valores de píxel y permite definir el tipo de interpolación y el modo de llenado de bordes en la imagen transformada.

10.8. Preguntas

10.8.1. Preguntas sobre hechos indicados en el texto

1. ¿Qué son las transformaciones afines y cómo se caracterizan?
2. ¿Qué técnica se emplea en el método de transformación inversa para asegurar que todas las posiciones de la imagen resultante reciban un valor asignado?
3. ¿Por qué se necesitan un total de ocho puntos para determinar una transformación proyectiva? Explique el proceso.
4. ¿Cuál es la función del parámetro `w` en una transformación proyectiva y de qué manera influye en las coordenadas resultantes?
5. ¿Si algunas transformaciones geométricas pueden realizarse con matrices de 2×2 , por qué se utiliza una representación con matrices de 3×3 ?

10.8.2. Preguntas de análisis y comprensión

1. ¿Cómo se puede utilizar una transformación afín para realizar una rotación y traslación simultáneas?
2. Comparar los efectos de una transformación de cizallamiento en la dirección `x` con una en la dirección `y`.
3. Explicar cómo una transformación geométrica puede corregir distorsiones en imágenes capturadas por cámaras.
4. Describe el método de transformación inverso y su utilidad en la visión por computadora.
5. ¿Cómo afecta la elección del método de interpolación (vecino más cercano vs. interpolación bilineal) a la calidad de la imagen resultante después de una transformación? Proporcione ejemplos de situaciones donde cada método sería más apropiado.

10.8.3. Ejercicios numéricos

1. Dada la matriz H de transformación afín de 3×3 , explique el significado de cada uno de los términos de la matriz.

$$H = \begin{bmatrix} 0.5 & 0.1 & 2 \\ 0.3 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Para la imagen I de 3×3 píxeles, realiza una translación de la imagen I con $tx = 2$ y $ty = 1$. Calcula las nuevas posiciones de los píxeles usando el método inverso por el vecino más cercano.

$$I = \begin{bmatrix} 3 & 5 & 2 \\ 8 & 1 & 9 \\ 7 & 6 & 4 \end{bmatrix}$$

3. Realiza el escalamiento de la imagen I de 2×2 píxeles con un factor $Sx = 2$ y $Sy = 1.5$.

$$I = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

4. A partir de una imagen original I y realiza el cálculo de las coordenadas para una rotación de 45 grados usando el método inverso por el vecino más cercano.

$$I = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

5. Considere los siguientes cuatro pares de puntos correspondientes:

$$\begin{array}{ll} p_1 = (0, 0), & p'_1 = (1, 1) \\ p_2 = (2, 0), & p'_2 = (3, 1) \\ p_3 = (2, 2), & p'_3 = (3, 3) \\ p_4 = (0, 2), & p'_4 = (1, 3) \end{array}$$

Determine la matriz de transformación proyectiva H que mapea estos puntos, mostrando los pasos para configurar el sistema de ecuaciones y resuelva para los elementos de H .

Capítulo 11

Operaciones Morfológicas Binarias

La morfología matemática es una técnica con base en la teoría de conjuntos y en la topología que se utiliza para extraer información sobre la forma y estructura de los objetos. Esta es altamente utilizada en el procesamiento de imágenes para analizar y manipular formas geométricas, lo que permite que sea utilizada comúnmente para extraer características útiles de una imagen, como bordes, contornos, regiones, áreas de interés entre otros usos. Las dos operaciones morfológicas básicas son la dilatación y la erosión. La primera consiste en expandir el área de los objetos de la imagen, mientras que la erosión reduce el área de los objetos. Ambas operaciones se realizan utilizando un elemento estructurante, el cual es una matriz binaria que define el área a expandir o reducir dentro de una vecindad para cada pixel. La morfología matemática se puede aplicar a imágenes binarias o en escala de grises. En el caso de las primeras, se definen de forma natural en términos de conjuntos. Por lo anterior, las binarias son más comunes, ya que las operaciones se pueden definir de manera más clara de forma binaria [11, 13, 53, 54].

11.1. Conceptos Básicos de Teoría de Conjuntos y Lógica Binaria

La teoría de conjuntos es una rama fundamental de las matemáticas que estudia las colecciones de objetos, llamados **elementos**. Los conceptos de unión, intersección y complemento son operaciones básicas que permiten combinar y relacionar conjuntos de manera formal.

La estrecha relación entre la teoría de conjuntos y la lógica binaria es fundamental en la morfología matemática, que es una herramienta clave en el procesamiento de imágenes. Por su parte, la matemática utiliza las operaciones de conjuntos (unión, intersección, complemento) y sus correspondientes operadores lógicos (OR, AND, NOT) para analizar y procesar estructuras dentro de las imágenes digitales que se modelan como conjuntos de píxeles, permitiendo así modificar y extraer información relevante de las regiones, facilitando tareas como la detección de bordes, la segmentación y la eliminación de ruido. La Fig. 11.1 ilustra de manera conceptual las operaciones de conjuntos y su relación con las lógicas.

11.1.1. Unión y el OR Lógico

La **unión** de dos conjuntos A y B es la región que contiene todos los elementos que pertenecen a A , a B . Se denota por $A \cup B$.

$$A \cup B = \{x \mid x \in A \text{ o } x \in B\}$$

Esto significa que un elemento x está en $A \cup B$ si x pertenece a A **o** pertenece a B (incluyendo el caso en que pertenece a ambos).

En lógica proposicional, el **OR** (disyunción) es una operación binaria que resulta verdadera si al menos una de las

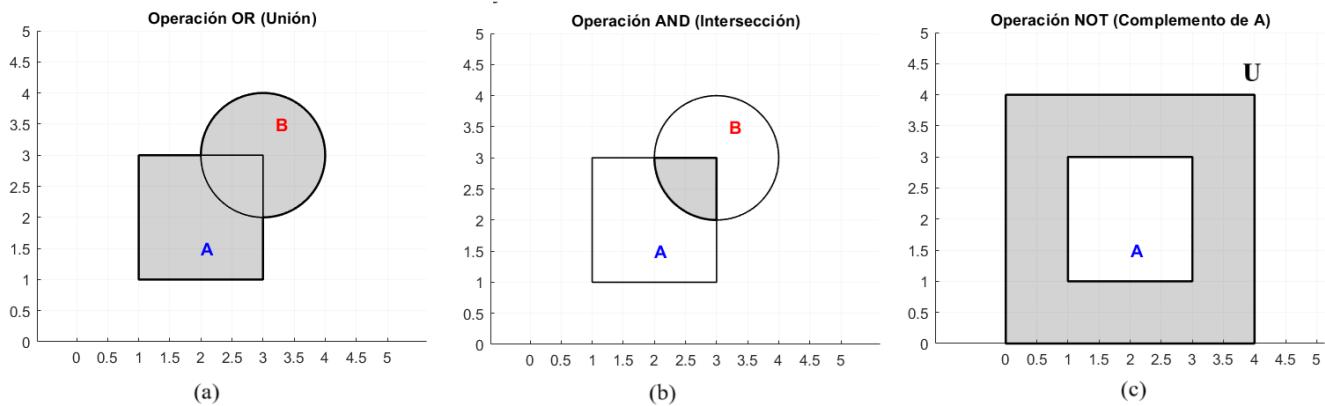


Figura 11.1: Explicación conceptual de algunas operaciones de conjuntos y lógicas básicas. (a) Unión y OR lógico, (b) Intersección y AND lógico y (c) Complemento y NOT lógico.

proposiciones es verdadera.

Si p y q son proposiciones lógicas, entonces:

$$p \vee q = \begin{cases} \text{Verdadero,} & \text{si } p \text{ es verdadero o } q \text{ es verdadero (o ambos)} \\ \text{Falso,} & \text{si } p \text{ y } q \text{ son falsos} \end{cases}$$

La unión de conjuntos y el operador lógico OR son análogos porque ambos combinan elementos o proposiciones que cumplen al menos una de las condiciones. Si se consideran conjuntos de valores de verdad, la unión corresponde al OR lógico. Véase la Fig(a).11.1.

11.1.2. Intersección y AND Lógico

La **intersección** de dos conjuntos A y B es la región que contiene todos los elementos que son comunes a ambos conjuntos. Se denota por $A \cap B$.

$$A \cap B = \{x \mid x \in A \text{ y } x \in B\}$$

Un elemento x está en $A \cap B$ si x pertenece tanto a A como a B .

En lógica proposicional, el **AND** (conjunción) es una operación binaria que resulta verdadera solo si ambas proposiciones son verdaderas.

Si p y q son proposiciones lógicas, entonces:

$$p \wedge q = \begin{cases} \text{Verdadero,} & \text{si } p \text{ es verdadero y } q \text{ es verdadero} \\ \text{Falso,} & \text{en cualquier otro caso} \end{cases}$$

La intersección de conjuntos y el operador lógico AND son análogos porque ambos requieren que se cumplan simultáneamente dos condiciones. Al igual que en la unión, si se consideran conjuntos de valores de verdad, la intersección corresponde al AND lógico. Véase la Fig(b).11.1.

11.1.3. Complemento y el NOT Lógico

El complemento de un conjunto A respecto a un conjunto universal U es el conjunto de todos los elementos que están en U pero no en A . Se denota por A' o $U \setminus A$.

$$A' = U \setminus A = \{x \mid x \in U \text{ y } x \notin A\}$$

Este se representa como todo lo que no está en A dentro del universo considerado.

En lógica proposicional, el NOT (negación) es una operación unaria que invierte el valor de verdad de una proposición.

Si p es una proposición lógica, entonces:

$$\neg p = \begin{cases} \text{Verdadero,} & \text{si } p \text{ es falso} \\ \text{Falso,} & \text{si } p \text{ es verdadero} \end{cases}$$

El complemento de un conjunto y el operador lógico NOT son análogos porque ambos consideran la negación de una condición: en conjuntos, los elementos que no pertenecen a A ; en lógica, el valor de verdad opuesto de p . Véase la Fig(c).11.1.

11.2. El Elemento Estructurante

El elemento estructurante (S) resulta ser una plantilla de ceros y unos que define el vecindario de cada pixel de la imagen a operar en relación con algún origen sobre la máscara. Este se recorre sobre toda la imagen binaria a transformar buscando las coincidencias entre pixeles de la imagen y el elemento estructurante (S). El origen del S no tiene que estar en su centro, pero a menudo lo está. El elemento estructurante puede tener diferentes formas y tamaños, como un rectángulo, un disco, una cruz o una línea, siendo su elección dependiente de la aplicación específica y de los objetivos del procesamiento de la imagen [13, 53].

A continuación, se presenta una descripción del uso de cada elemento estructurante en morfología matemática para el procesamiento de imágenes:

11.2.1. Elemento Estructurante Cuadrado

Es un elemento estructurante de forma cuadrada que se utiliza para erosionar y dilatar bordes rectangulares. Al elegir un elemento estructurante cuadrado del tamaño apropiado, se puede lograr la operación morfológica de manera uniforme en todas las direcciones. En muchos casos, el elemento estructurante cuadrado es una adecuada elección porque es fácil de definir y puede ser utilizado en una amplia variedad de situaciones sin comprometer dramáticamente la forma.

11.2.2. Elemento Estructurante Rectángulo

Este el elemento tiene forma asimétrica y su orientación puede afectar el resultado de la operación morfológica. Por lo tanto, cuando se utiliza, las operaciones morfológicas pueden producir resultados que no son simétricos y que pueden variar en función de la orientación del elemento. En conclusión, se puede usar cuando se desea favorecer una orientación específica de bordes rectangulares.

11.2.3. Elemento Estructurante Disco

Se recomienda cuando se desea realizar una operación que tenga en cuenta todos los pixeles que se encuentran a una distancia determinada de un píxel en particular, sin importar la orientación, y que además tenga una forma

Elementos Estructurantes

a) Todo lleno				
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

b) Rectángulo				
0	0	0	0	0
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	0	0	0	0

c) Disco				
0	1	1	1	0
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	1	1	1	0

d) Cruz				
0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

e) Linea horizontal				
0	0	0	0	0
0	0	0	0	0
1	1	1	1	1
0	0	0	0	0
0	0	0	0	0

f) Diamante				
0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

Figura 11.2: Ejemplos de Elementos Estructurante. a) Cuadrado, b) Rectangular, c) Disco, d) Cruz, e) Lineal y f) Diamante.

suave y redondeada. Es útil en muchos casos en los que se desea realizar una operación de dilatación o erosión suave y simétrica.

11.2.4. Elemento Estructurante Cruz

El elemento estructurante de forma de cruz sirve para la detección y el análisis de estructuras en forma de cruz. Debido a su forma, puede no ser adecuado para producir bordes más complejos o irregulares. No se considera adecuado para ciertas operaciones morfológicas, como el cierre, que requiere un elemento estructurante que cubra completamente el objeto que se está procesando. Un elemento estructurante demasiado pequeño puede no tener suficiente alcance para producir el efecto deseado, mientras que un elemento estructurante demasiado grande puede tener un efecto excesivo y distorsionar la imagen.

11.2.5. Elemento Estructurante Línea

El elemento estructurante de forma lineal se emplea para la detección y el análisis de estructuras lineales. Se recomienda su uso cuando se desea realizar una operación de erosión o apertura para eliminar objetos delgados en una imagen binaria, como líneas finas, mientras se conservan objetos más gruesos.

11.3. La Erosión

La erosión es una operación morfológica utilizada en el procesamiento de imágenes para reducir el tamaño de los objetos en una imagen. La erosión implica comparar el elemento estructurante con cada píxel de la imagen que se explora; si todos los píxeles cubiertos por el elemento estructurante son blancos (1), entonces el píxel central del elemento estructurante se mantiene blanco (1). De lo contrario, el píxel central coincidente con la imagen se vuelve negro (0). Esto tiene el efecto de reducir el tamaño de los objetos en la imagen. La Figura 11.3 ilustra el proceso de Erosión desde una imagen en escala de grises [13, 53].

La erosión del conjunto I por el elemento estructurante S se denota como:

$$I_e = I \ominus S = \{p \in \mathbb{Z}^2 \mid S_p \subseteq I\}$$

donde:

- I_e es la imagen erosionada resultante.
- \ominus denota la operación de erosión.
- S_p es la traslación del elemento estructurante S por el vector p :

$$S_p = \{s + p \mid s \in S\}$$

$s + p$ representa la suma vectorial, trasladando cada punto s de S a una nueva posición en I .

11.4. Algoritmo de Erosión

1. Definir la imagen de entrada I y el elemento estructurante S .
2. Inicializar la imagen de salida erosionada I_e como una imagen en blanco.
3. Explorar cada píxel de la imagen de entrada:

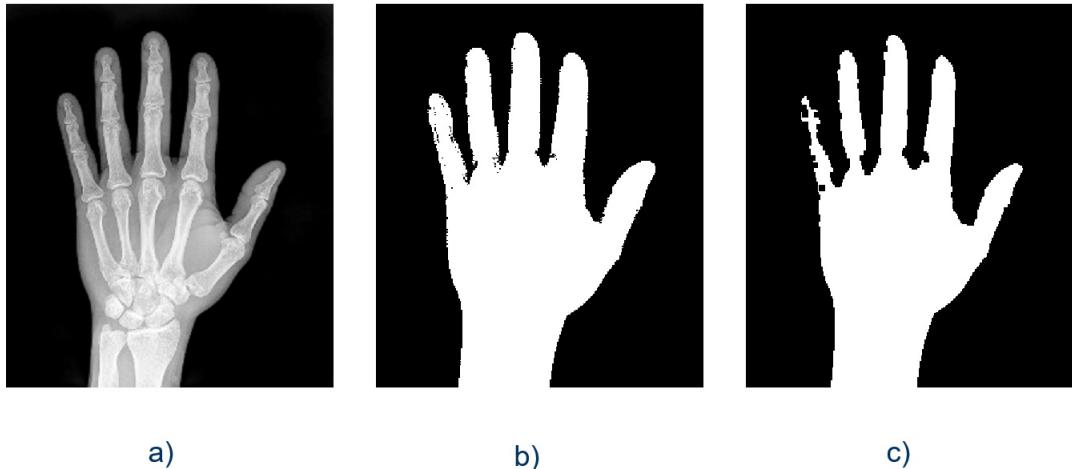


Figura 11.3: Proceso de Erosión morfológica. a) Imagen en escala de grises, b) Imagen Binarizada y c) Imagen Erosionada.

- a) Colocar el elemento estructurante S en el píxel actual p de la imagen.
 - b) Comparar el elemento estructurante con los píxeles de la imagen cubiertos por el elemento.
 - c) Si *todos* los píxeles cubiertos por el elemento estructurante S son blancos (1), el píxel central se mantiene blanco (1) en la imagen de salida I_e (Operación AND).
 - d) De lo contrario, el píxel central se vuelve negro en la imagen de salida.
4. Repetir el paso 3 para cada píxel de la imagen de entrada.
 5. Devolver la imagen de salida como resultado.

Es importante tener en cuenta que la erosión se puede realizar varias veces en una imagen para reducir aún más el tamaño de los objetos. También se pueden utilizar diferentes formas y tamaños de elementos estructurantes para lograr diferentes efectos de erosión en la imagen. La Fig.11.4 muestra un ejemplo de erosión de una región con un elemento estructurante en cruz.

11.5. La Dilatación

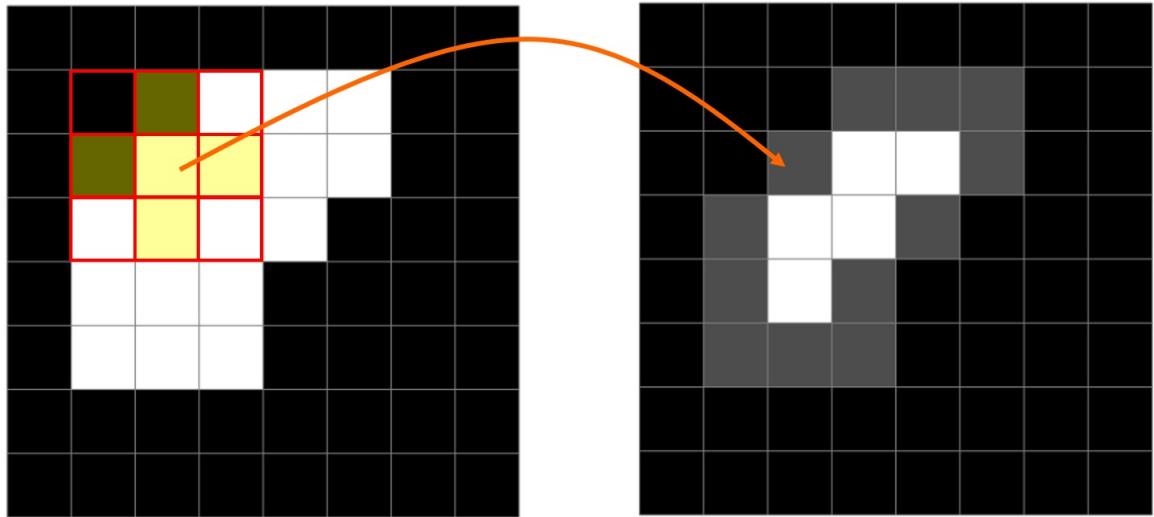
La dilatación es otra operación morfológica utilizada en el procesamiento de imágenes que tiene el efecto contrario a la erosión, es decir, expande el tamaño de los objetos en una imagen. Esta se realiza mediante la exploración de la imagen con el elemento estructurante. El proceso de dilatación implica comparar el elemento estructurante con cada píxel de la imagen y, si al menos un píxel cubierto por el elemento estructurante es blanco (1), el píxel central se mantiene blanco (1). De lo contrario, el píxel central coincidente con la imagen se vuelve negro. Esto tiene el efecto de expandir el tamaño de los objetos en la imagen. La Figura 11.5 ilustra el proceso de Dilatación desde una imagen en escala de grises [13, 53].

La dilatación del conjunto I por el elemento estructurante S se denota como:

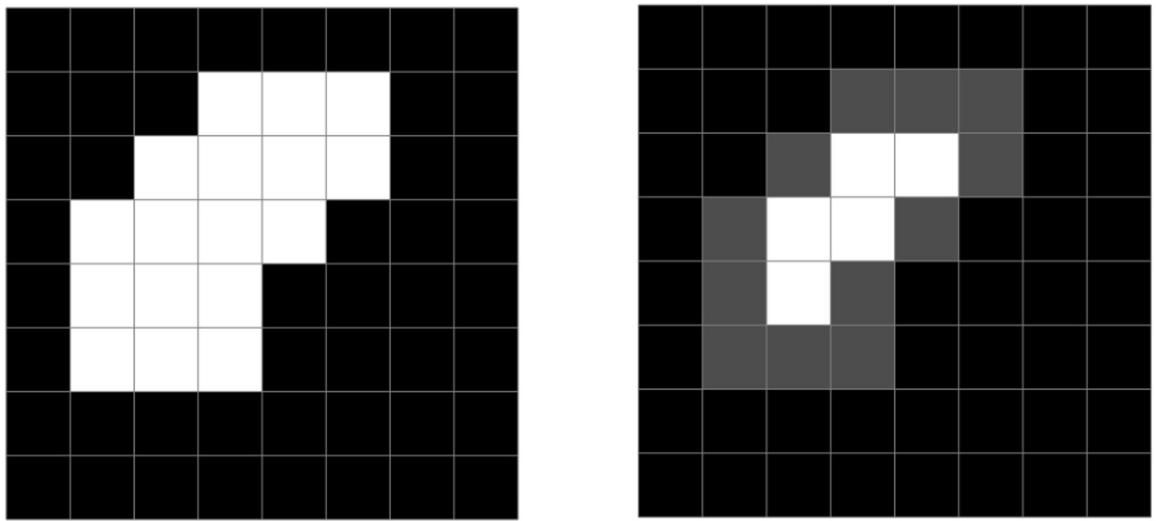
$$I_d = I \oplus S = \{p \in \mathbb{Z}^2 \mid S_p \cap I \neq \emptyset\}$$

donde:

- I_d es la imagen dilatada resultante.



(a) Proceso de erosión ilustrado para un pixel. Notese que como todo el elemento estructurante no cabe en la parte blanca (1), entonces el pixel central debe tornar negro (0).



(b) Proceso de erosión completado para todos los pixeles.

Figura 11.4: Ejemplo de Erosión. (a) para un pixel y (b) paara todo los pixeles.

- \oplus denota la operación de dilatación.
- S_p es la traslación del elemento estructurante S por el vector p :

$$S_p = \{s + p \mid s \in S\}$$

$s + p$ representa la suma vectorial, trasladando cada punto s de S a una nueva posición en I .

11.5.1. Algoritmo de Dilatación

1. Definir la imagen de entrada I y el elemento estructurante S .
 2. Inicializar la imagen de salida Id como una imagen en blanco.
- a) Explorar cada píxel de la imagen de entrada:

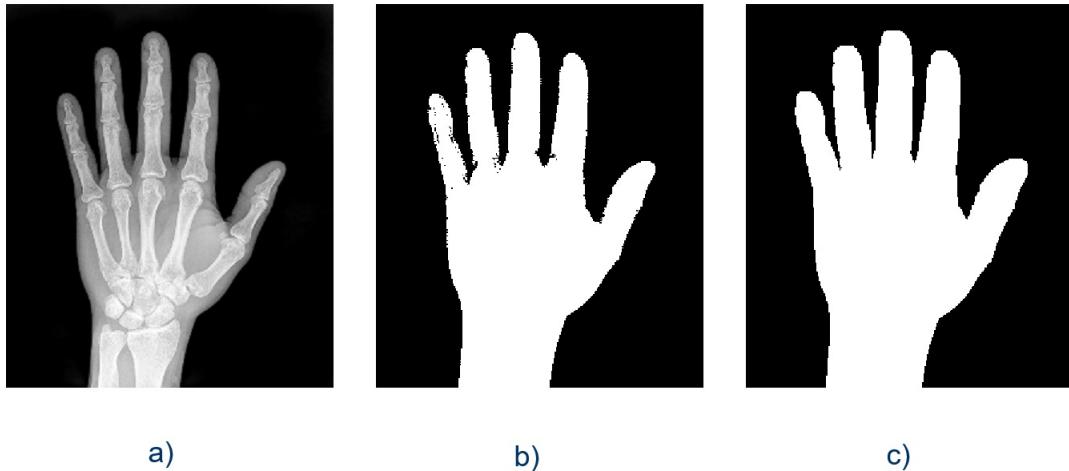


Figura 11.5: Proceso de Dilatación morfológica. a) Imagen en escala de grises, b) Imagen Binarizada y c) Imagen Dilatada.

- b) Colocar el elemento estructurante S en el píxel actual p de la imagen.
 - c) Comparar el elemento estructurante con los píxeles de la imagen cubiertos por el elemento.
 - d) Si al menos *un píxel* cubierto por el elemento estructurante S es blanco (1), el píxel central se mantiene blanco (1) en la imagen de salida I_d (Operación OR).
 - e) De lo contrario, el píxel central se vuelve negro en la imagen de salida.
3. Repetir el paso 3 para cada píxel de la imagen de entrada.
 4. Devolver la imagen de salida como resultado.

Es importante tener en cuenta que la dilatación también se puede realizar varias veces en una imagen para expandir aún más el tamaño de los objetos. Además, al igual que con la erosión, se pueden utilizar diferentes formas y tamaños de elementos estructurantes para lograr diferentes efectos sobre la imagen. La Fig.11.6 muestra un ejemplo de dilatación con un elemento estructurante en cruz.

11.6. Dualidad entre Erosión y Dilatación

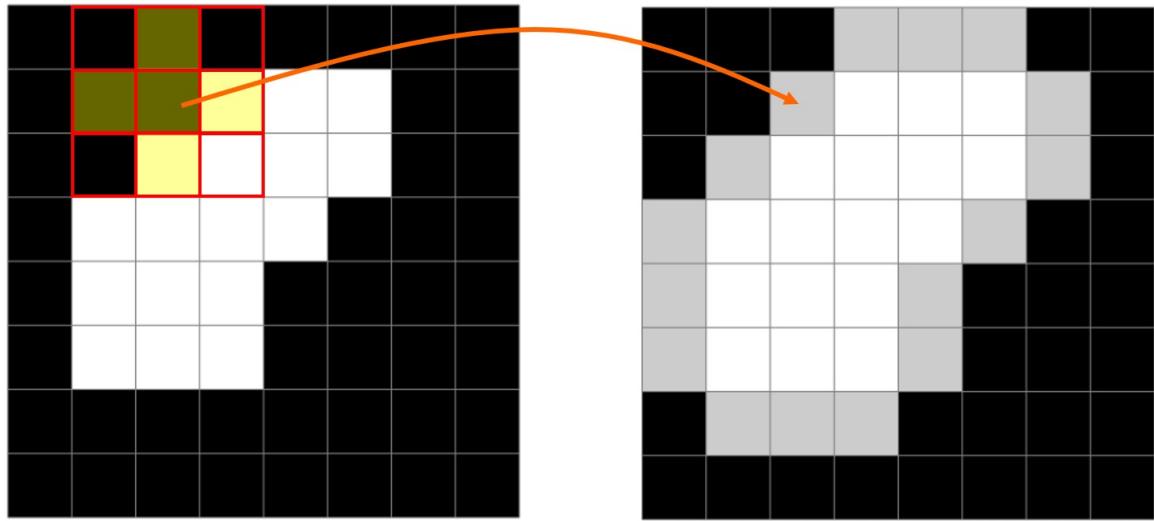
En morfología matemática, las operaciones de **erosión** y **dilatación** están estrechamente relacionadas y se consideran duales entre sí respecto a la complementación y, en algunos casos, la reflexión del elemento estructurante S . Esta dualidad es fundamental para comprender cómo estas operaciones interactúan y cómo se pueden derivar una de la otra bajo ciertas condiciones. La dualidad entre la erosión y la dilatación se expresa mediante las siguientes ecuaciones:

$$I \ominus S = (I^c \oplus S^s)^c$$

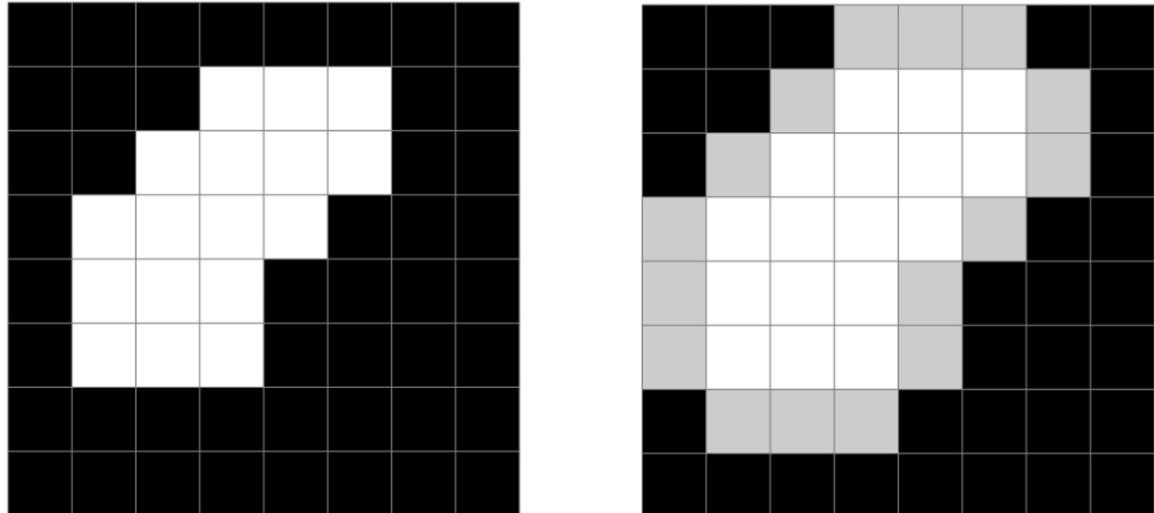
$$I \oplus S = (I^c \ominus S^s)^c$$

donde:

- I es la imagen de entrada.



(a) Proceso de dilatación para un pixel. Nótese que el elemento estructurante alcanza a tocar un pixel blanco (1), por lo que el centro se lleva a blanco (1) también.



(b) Proceso de dilatación completado para todos los píxeles.

Figura 11.6: Ejemplo de Dilatación. (a) para un pixel y (b) para todo los píxeles.

- I^c es el complemento de I :

$$I^c = \mathbb{Z}^2 \setminus I$$

- S^s es la reflexión de S respecto al origen:

$$S^s = \{-s \mid s \in S\}$$

- \ominus denota la operación de erosión.

- \oplus denota la operación de dilatación.

La dualidad muestra que la erosión de una imagen I por un elemento estructurante S es equivalente al complemento de la dilatación del complemento de I por la reflexión de S , y viceversa.

La reflexión del elemento estructurante S respecto al origen se define como:

$$S^s = \{-s \mid s \in S\}$$

Esto implica que cada punto del elemento estructurante se refleja respecto al origen, es decir, se invierten sus coordenadas. La necesidad de reflejar el elemento estructurante S depende de cómo se definan las operaciones de erosión y dilatación en el contexto específico.

En muchas aplicaciones prácticas y definiciones, especialmente en el procesamiento digital de imágenes, se asume que el elemento estructurante S es simétrico respecto al origen (por ejemplo, un cuadrado centrado en el origen), por lo que la reflexión de S no afecta el resultado ya que $S = S^s$; sin embargo, si el elemento estructurante no es simétrico, la reflexión S^s se vuelve necesaria para asegurar la dualidad exacta entre erosión y dilatación, dado que la posición relativa de S respecto al origen influye en cómo se aplican las operaciones morfológicas; por esta razón, algunos autores y bibliografías definen la dilatación y la erosión sin incluir la reflexión en sus fórmulas, especialmente cuando trabajan en espacios discretos y con elementos estructurantes simétricos, simplificando así las definiciones y permitiendo omitir la reflexión.

11.7. La Apertura Morfológica

La apertura (opening en inglés) es una operación morfológica que se utiliza en el procesamiento de imágenes para eliminar pequeños objetos y detalles en los bordes de los objetos más grandes. Consiste en aplicar primero una erosión a la imagen, seguida de una dilatación. La erosión reduce el tamaño de los objetos en la imagen y, por lo tanto, elimina los detalles más pequeños, mientras que la dilatación tiene el efecto de expandir los objetos nuevamente al tamaño original, pero sin los detalles que se eliminaron durante la erosión [13, 53]. La apertura ayuda a suavizar los bordes de los objetos en la imagen mientras se eliminan los detalles no deseados.

11.7.1. Algoritmo Apertura

1. Definir la imagen de entrada y el elemento estructurante.
2. Realizar una erosión en la imagen de entrada utilizando el elemento estructurante.
3. Realizar una dilatación en la imagen resultante de la erosión utilizando el mismo elemento estructurante.
4. Devolver la imagen resultante de la dilatación como resultado.

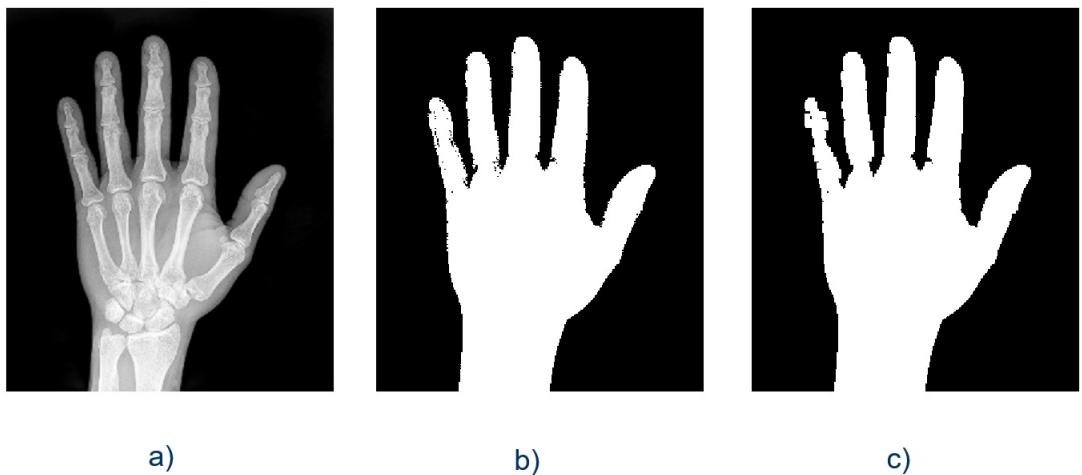
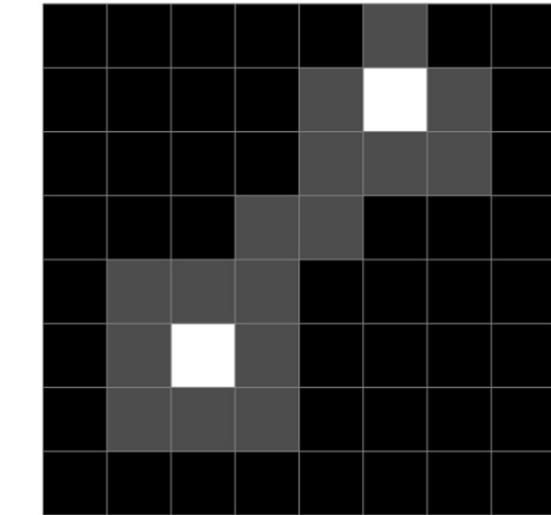
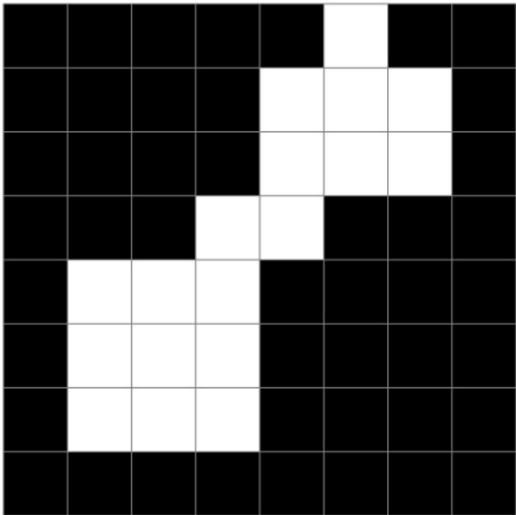
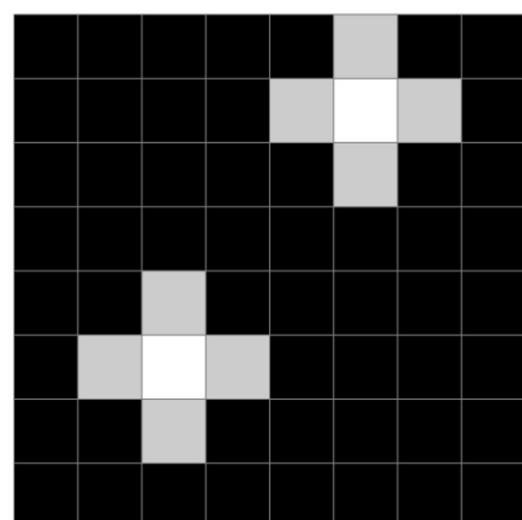
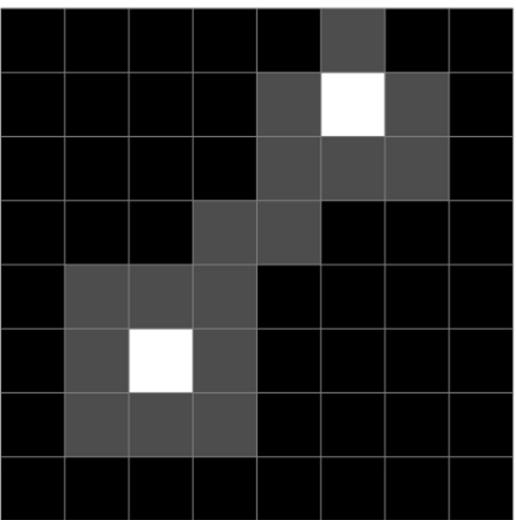


Figura 11.7: Proceso de Apertura morfológica. a) Imagen en escala de grises, b) Imagen Binarizada y c) Imagen con Apertura.

La apertura es una técnica muy útil en el procesamiento de imágenes, ya que ayuda a reducir el ruido y los detalles innecesarios en la imagen, mejorando así la calidad visual y la precisión de las operaciones que se realicen posteriormente. Esta separa sutilmente los objetos unidos manteniendo el tamaño general de los objetos.



(a)



(b)

Figura 11.8: Ejemplo de apertura. Ejemplo de Cerramiento. (a) Primero se realiza la erosión y (b) seguidamente la dilatación con el mismo elemento estructurante

11.8. El Cerramiento Morfológico

El cerramiento (closing en inglés) es una operación morfológica que se utiliza en el procesamiento de imágenes. A diferencia de la apertura, el cerramiento se utiliza para eliminar los agujeros pequeños y estrechos en los objetos y para cerrar brechas entre ellos. Consiste en aplicar primero una dilatación a la imagen, seguida de una erosión. La dilatación tiene el efecto de expandir los objetos en la imagen y cerrar las brechas, mientras que la erosión tiene el efecto de reducir los objetos al tamaño original y cerrar los agujeros pequeños [13, 53].

11.8.1. Algoritmo de Cerramiento

1. Definir la imagen de entrada y el elemento estructurante.
2. Realizar una dilatación en la imagen de entrada utilizando el elemento estructurante.
3. Realizar una erosión en la imagen resultante de la dilatación utilizando el mismo elemento estructurante.

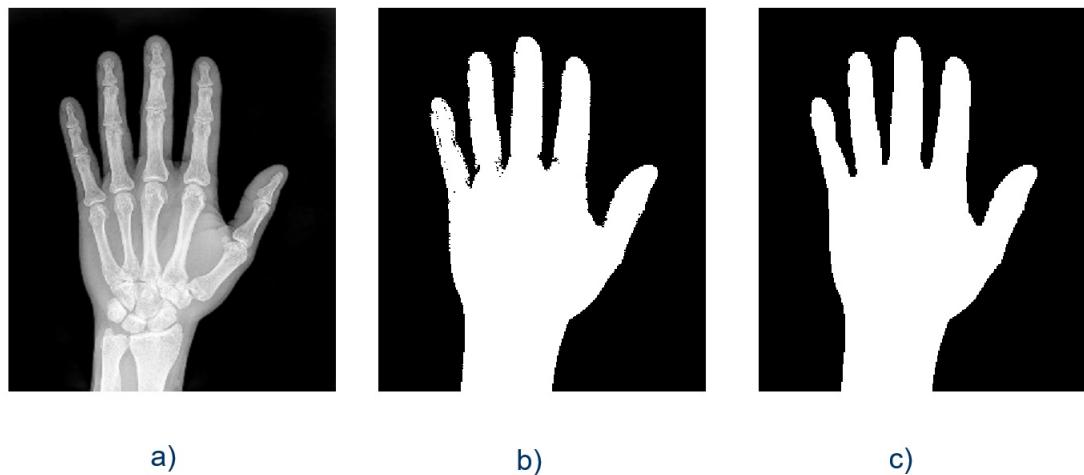


Figura 11.9: Proceso de Cerramiento morfológico. a) Imagen en escala de grises, b) Imagen Binarizada y c) Imagen con Cerramiento.

4. Devolver la imagen resultante de la erosión como resultado.

El cerramiento es útil en el procesamiento de imágenes para suavizar los bordes de los objetos y mejorar la conectividad entre los objetos en la imagen. Además, se puede utilizar para llenar pequeños agujeros y para suavizar la forma de los objetos en la imagen.

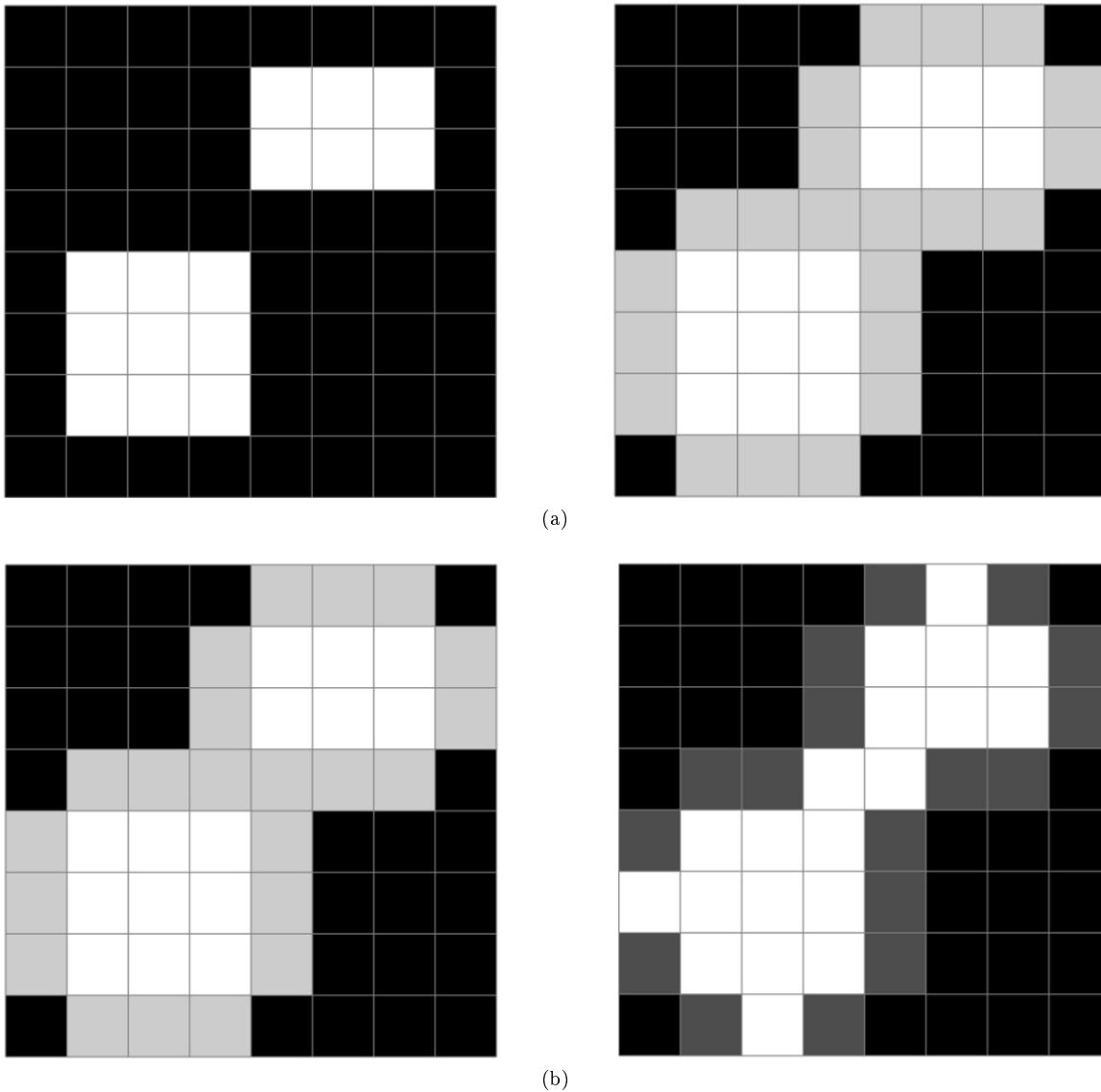


Figura 11.10: Ejemplo de Cerramiento. (a) Primero se realiza la dilatación y (b) seguidamente la erosión con el mismo elemento estructurante.

11.8.2. Ejemplo en Python: segmentación mediante morfología

El código desarrollado aplica operaciones morfológicas clásicas sobre una imagen real de herramientas, permitiendo ilustrar paso a paso el proceso de segmentación estructural. A partir de una imagen a color, se realiza la conversión a escala de grises, binarización automática mediante umbral adaptativo e inversión lógica para resaltar los objetos. Posteriormente, se aplican operaciones de dilatación y erosión (cierre morfológico) para consolidar las regiones correspondientes a las herramientas y eliminar imperfecciones. Finalmente, la imagen original se segmenta aplicando la máscara obtenida del cierre, aislando visualmente los objetos de interés.

Solución

```

1 # Importación de Librerías Básicas
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # Importar funciones personalizadas (ip_functions)
6 from ip_functions import *

```

```

7
8 # Abrir la imagen
9 Icolor = plt.imread("tools.jpg")
10 I = np.array(Icolor)
11
12 # Factor de Escala S
13 S = 1
14 Icolor = Icolor[0::S, 0::S, :]
15
16 # Convertir a escala de grises
17 Igray = rgb2gray(Icolor)
18
19 # Binarizar la imagen (inversión para que los objetos sean blancos)
20 T = graythresh(Igray)
21 Ibin = ~im2bw(Igray, T)
22
23 # Crear el elemento estructurante
24 EE = np.ones((5, 5)).astype('bool')
25
26 # Aplicar dilatación
27 Idil = imdilate(Ibin, EE)
28
29 # Aplicar erosión sobre la dilatada (cierre)
30 Icerrada = imerode(Idil, EE)
31
32 # Segmentar la imagen original con la máscara del cierre
33 Isegmentada = np.zeros_like(Icolor)
34 for i in range(3):
35     Isegmentada[:, :, i] = Icolor[:, :, i] * Icerrada.astype(np.uint8)
36
37 # Mostrar las 6 imágenes en un subplot 2x3
38 plt.figure(figsize=(15, 10))
39
40 plt.subplot(2, 3, 1)
41 plt.imshow(Icolor)
42 plt.title("a) Imagen Original a Color")
43 plt.axis('off')
44
45 plt.subplot(2, 3, 2)
46 plt.imshow(Igray, cmap='gray')
47 plt.title("b) Escala de Grises")
48 plt.axis('off')
49
50 plt.subplot(2, 3, 3)
51 plt.imshow(Ibin, cmap='gray')
52 plt.title("c) Imagen Binarizada")
53 plt.axis('off')
54
55 plt.subplot(2, 3, 4)
56 plt.imshow(Idil, cmap='gray')
57 plt.title("d) Imagen Dilatada")
58 plt.axis('off')
59
60 plt.subplot(2, 3, 5)
61 plt.imshow(Icerrada, cmap='gray')
62 plt.title("e) Imagen Cerrada (Cierre)")
63 plt.axis('off')
64
65 plt.subplot(2, 3, 6)
66 plt.imshow(Isegmentada)
67 plt.title("f) Imagen Segmentada")
68 plt.axis('off')
69
70 plt.tight_layout()
71 plt.show()

```

En la Fig.11.11 se muestran seis imágenes organizadas secuencialmente para mostrar el proceso completo de segmentación de objetos mediante operaciones morfológicas. La imagen (a) corresponde a la fotografía original a color, en la que se observan varias herramientas sobre un fondo claro. En (b) se presenta la conversión a escala

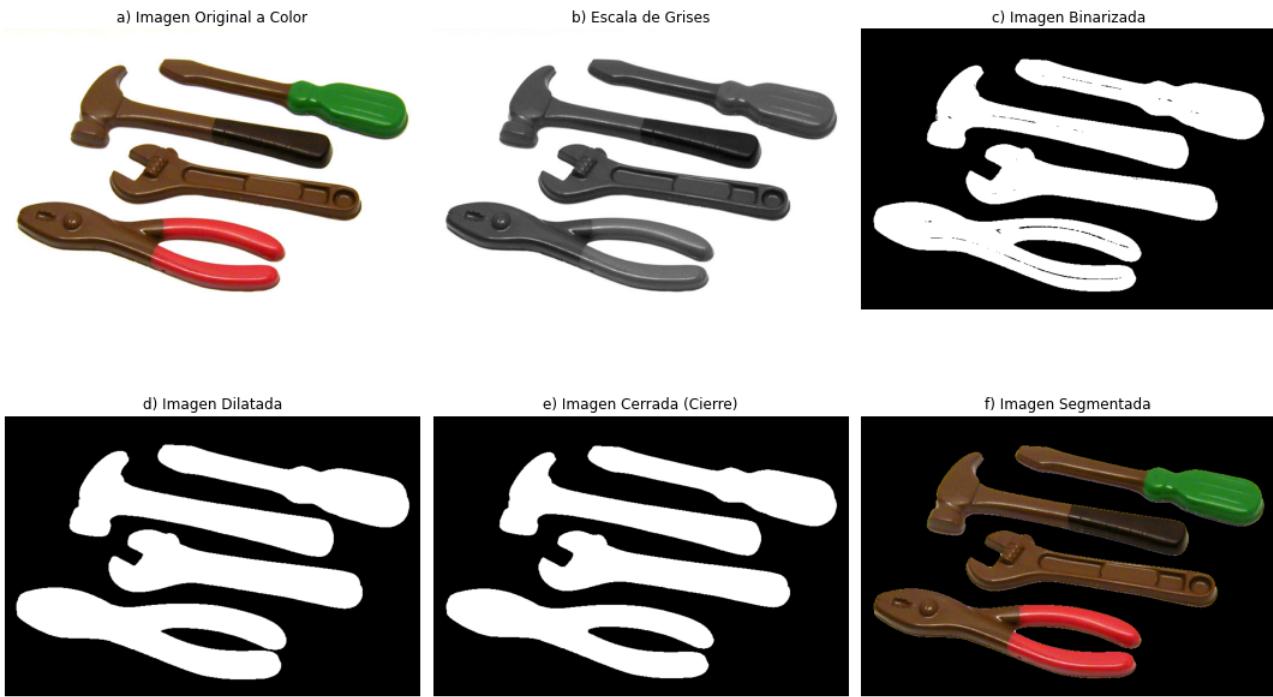


Figura 11.11: Proceso de segmentación por cierre morfológico: a) imagen original, b) escala de grises, c) binarización, d) dilatación, e) cierre (dilatación seguida de erosión), f) segmentación de herramientas aplicando la máscara cerrada sobre la imagen original.

de grises, paso previo fundamental para aplicar umbrales de segmentación por tonalidad. La imagen (c) muestra la versión binarizada mediante Otsu, donde se invierte la lógica para que los objetos aparezcan en blanco y el fondo en negro. La imagen (d) presenta el resultado de aplicar una dilatación, la cual expande las regiones blancas, cerrando pequeños huecos o discontinuidades. En (e), se aplica una erosión sobre la imagen dilatada, completando la operación de cierre morfológico, que permite separar las regiones correspondientes a las herramientas. Finalmente, en (f), se muestra la segmentación obtenida al aplicar la máscara cerrada sobre la imagen original, permitiendo aislar visualmente solo las herramientas del fondo. Las funciones referidas allí, se encuentran en el capítulo 13.

11.9. Funciones

Funciones morfológicas utilizadas

Funciones Morfológicas Binarias en MATLAB

Las funciones morfológicas en MATLAB se aplican a imágenes binarias utilizando un **elemento estructurante EE**, el cual se define mediante la función `strel`. Este elemento determina la forma y tamaño de la vecindad que se utiliza para modificar la imagen. Las funciones más comunes incluyen:

1. `imerode(I, EE)`: Realiza la operación de **erosión** sobre la imagen binaria `I`, eliminando píxeles en los bordes de los objetos. El píxel central se conserva solo si todos los píxeles bajo el elemento estructurante `EE` están activos (1). Esta operación es útil para eliminar detalles finos, separar objetos conectados por puntos estrechos o reducir el tamaño de las regiones blancas.
2. `imdilate(I, EE)`: Aplica la operación de **dilatación**, expandiendo las regiones blancas de la imagen binaria `I`. Se agrega un píxel blanco (1) en la posición central si al menos un píxel bajo el elemento estructurante `EE` es blanco. Esta operación permite llenar huecos pequeños, cerrar discontinuidades y agrandar los objetos.
3. `imclose(I, EE)`: Ejecuta el **cierre morfológico**, que consiste en una dilatación seguida de una erosión con el mismo `EE`. Es especialmente útil para llenar huecos o espacios pequeños dentro de los objetos, conectar

regiones próximas sin alterar significativamente la forma de los objetos originales.

4. `imopen(I, EE)`: Realiza la **apertura morfológica**, que implica una erosión seguida de una dilatación con el mismo EE. Este proceso elimina pequeñas regiones o ruidos aislados en la imagen, manteniendo intacta la forma general de los objetos principales.

Definición del Elemento Estructurante `strel`

El elemento estructurante EE se define mediante la función `strel` de MATLAB. Algunos ejemplos comunes incluyen:

- `strel('square', n)`: genera un elemento estructurante cuadrado de tamaño $n \times n$.
- `strel('disk', r)`: genera un disco de radio r , útil para operaciones simétricas.
- `strel('line', len, angle)`: crea una línea recta de longitud `len` y ángulo `angle` en grados, ideal para estructuras orientadas.
- `strel('diamond', r)`: produce una forma de diamante con radio r desde el centro hasta los bordes.
- `strel('rectangle', [m n])`: define un rectángulo de dimensiones $m \times n$, útil para direcciones específicas.

La elección del `strel` depende de la forma del objeto a analizar y del tipo de transformación deseada. Su correcta selección mejora significativamente los resultados de segmentación y preprocesamiento en imágenes binarizadas.

11.10. Preguntas

Preguntas sobre hechos indicados en el texto

1. ¿Qué es la morfología matemática y en qué área de estudio se utiliza principalmente?
2. ¿Cuál es la función del elemento estructurante en las operaciones de morfología matemática?
3. ¿Qué condición debe cumplirse para que un píxel central se mantenga blanco (1) durante una operación de erosión?
4. ¿Cuáles son las operaciones morfológicas fundamentales descritas en el texto?
5. ¿Qué efecto tiene la operación de dilatación sobre los objetos de una imagen binaria?

Preguntas de análisis y comprensión con base en el texto

1. Explica por qué la morfología matemática es una herramienta útil en el procesamiento de imágenes binarias.
2. ¿Cuál es la relación entre los operadores de lógica binaria (AND, OR, NOT) y las operaciones de conjuntos aplicadas en morfología matemática?
3. Describe la importancia de aplicar la reflexión al elemento estructurante S cuando este no es simétrico respecto al origen.
4. Explica cómo se combinan las operaciones de erosión y dilatación en el cerramiento morfológico y cuál es la utilidad de esta combinación.
5. ¿Qué ventajas ofrece la operación de erosión cuando se aplica a una imagen que contiene ruido?

Ejercicios Numéricos

1. Se propone realizar una erosión directa y por dualidad sobre una imagen binaria representada por una matriz I de 10×10 , utilizando un elemento estructurante S cuadrado de 3×3 . Hacer el paso a paso a mano. Véase la Figura 11.12.

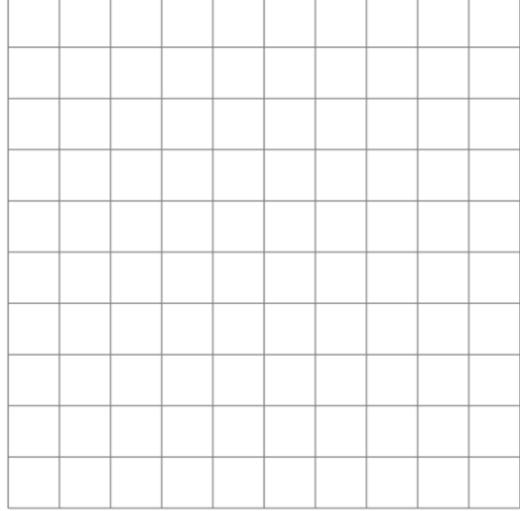
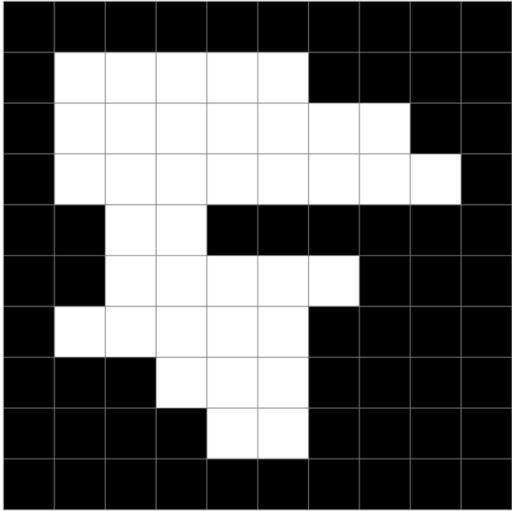


Figura 11.12: Plantilla para realizar la Erosión del ejercicio 1. Para efectos de facilidad en la visualización, se han dejado los 1 de color negro y los blancos como 0 en la gráfica.

2. Realizar una dilatación directa y por dualidad sobre una imagen binaria representada por una matriz I de 10×10 , utilizando un elemento estructurante S en línea de 1×3 . Hacer el paso a paso a mano. Véase la Figura 11.13.

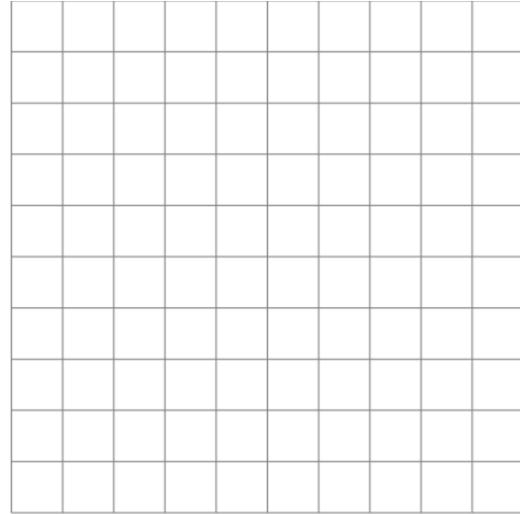
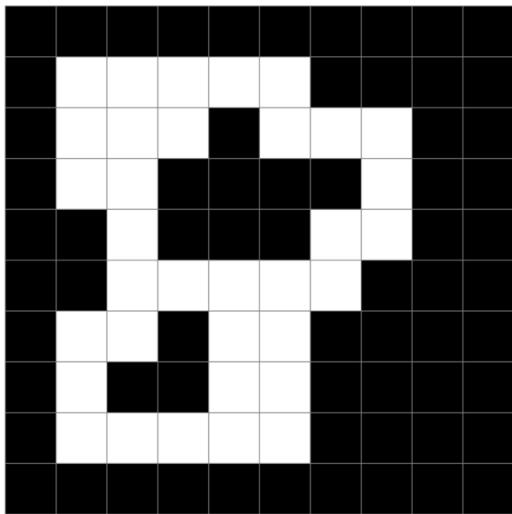


Figura 11.13: Plantilla para realizar la Dilatación del ejercicio 2. Para efectos de facilidad en la visualización, se han dejado los 1 de color negro y los blancos como 0 en la gráfica.

3. Sobre el ejercicio enunciado en el punto 1, realizar apertura con el mismo elemento estructurante y mostrar sus resultados. Hacer el paso a paso a mano usando la plantilla general mostrada en la Fig.11.15.

4. Sobre el ejercicio enunciado en el punto 2, realizar cerramiento con el mismo elemento estructurante y mostrar sus resultados. Hacer el paso a paso a mano usando la plantilla general mostrada en la Fig.11.15.

5. Se propone realizar la detección de bordes en una imagen binaria I de 10×10 que contiene un círculo aproximado en el centro. Hacer el paso a mano con un elemento estructurante S 3×3 usando la plantilla general mostrada en la Fig.11.14.

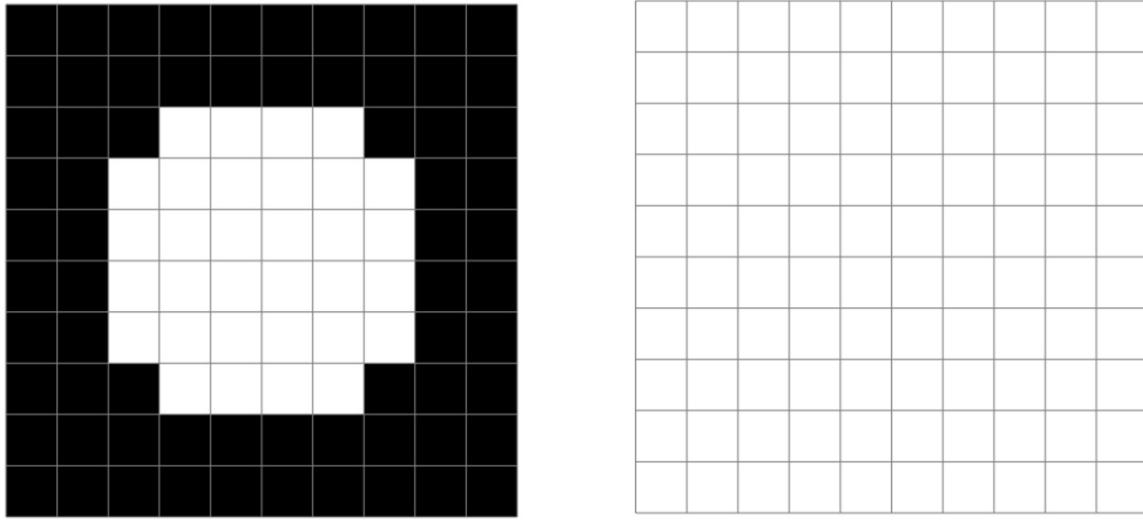


Figura 11.14: Imagen circular para ejercicio de referencia de detección de bordes.

- Realiza una erosión sobre la imagen I utilizando el elemento estructurante S , y luego resta la imagen erosionada de la imagen original. Esto resaltarán los píxeles en los bordes internos de la figura. La operación para el borde interno es:

$$\text{Borde_interno} = I - \text{Erosión}(I, S)$$

- Realiza una dilatación sobre la imagen I utilizando el elemento estructurante S , y luego resta la imagen original de la imagen dilatada. Esto resaltarán los píxeles en los bordes externos de la figura. La operación para el borde externo es:

$$\text{Borde_externo} = \text{Dilatación}(I, S) - I$$

- Visualiza los resultados obtenidos para el borde interno y el borde externo de la figura. Compara ambas imágenes para analizar cómo se destacan los contornos de cada caso.

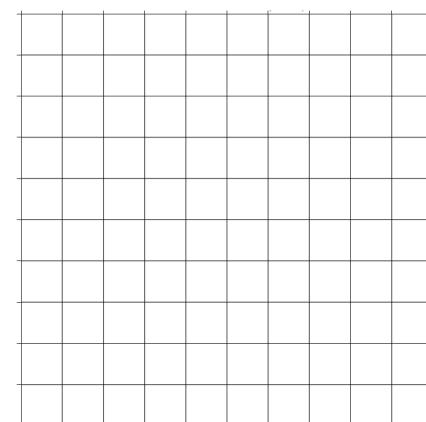
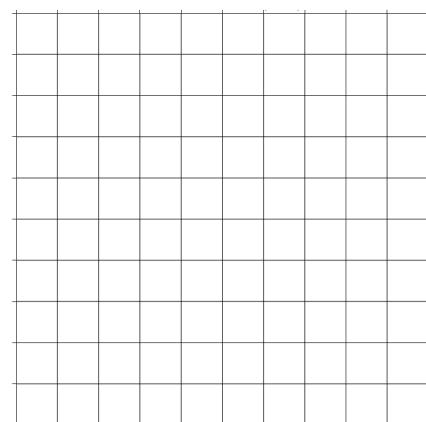
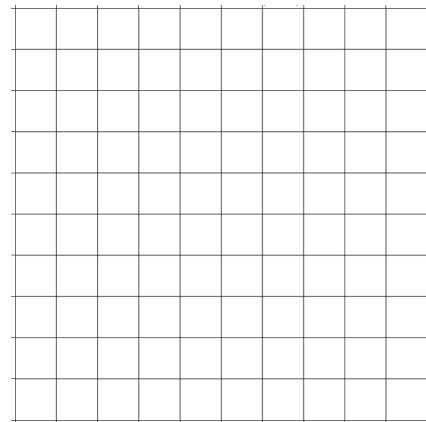
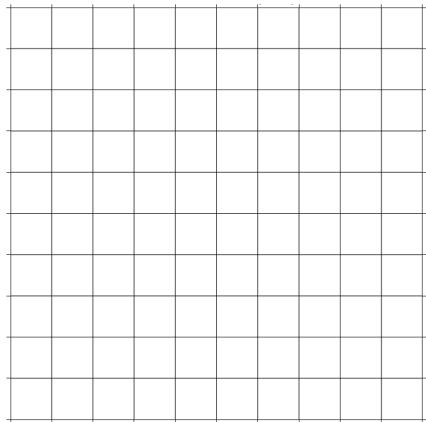


Figura 11.15: Plantilla general para realizar ejercicios de operaciones morfológicas.

Capítulo 12

Etiquetado

El etiquetado de componentes conectados es una de las operaciones más fundamentales en el procesamiento de imágenes binarias, puesto que se usa para extraer características estructurales o funcionales de los objetos presentes en una imagen digital. Como resultado de este proceso, se obtiene una imagen binaria en la que los objetos aparecen bien definidos. El objetivo del proceso es identificar cada región conexa, es decir, cada conjunto de píxeles conectados entre sí y asignar a todos los píxeles de dicha región una misma etiqueta, representada por un número entero único. Este procedimiento permite individualizar objetos, facilitando su análisis posterior, como el conteo, la medición de propiedades o la clasificación morfológica [6, 13, 16, 46].

12.1. Análisis de componente conexos

Se denomina componente conexo a un conjunto de píxeles en una imagen binaria que están conectados entre sí de acuerdo con un criterio de vecindad definido, como la conectividad 4 o 8. Dos píxeles pertenecen al mismo componente conexo si existe un camino formado exclusivamente por píxeles del objeto que los une mediante pasos consecutivos entre vecinos. En otras palabras, todos los píxeles de un componente conexo forman una región continua, sin interrupciones, dentro de la imagen.

12.2. Conectores bidimensionales

El *análisis de componentes conexos* (CCA, por sus siglas en inglés: *Connected Component Analysis*) tiene por objetivo principal identificar, etiquetar y aislar regiones que representan objetos completos o partes estructuradas de una escena. Este análisis se aplica principalmente a imágenes binarias, las cuales provienen de un proceso de segmentación que permite separar los objetos del fondo [6, 13]. La importancia del análisis de componentes conexos también radica en que transforma la imagen binaria en una representación estructurada, donde cada región conexa puede ser tratada como una unidad individual. A través del etiquetado de estas regiones, es posible realizar diversas tareas de análisis morfológico y cuantitativo, tales como:

- Contar cuántos objetos distintos existen en la imagen.
- Medir propiedades geométricas de cada objeto, como el área, el perímetro, la forma o la orientación.
- Filtrar componentes en función de su tamaño o forma, eliminando ruido o preservando solo objetos relevantes.
- Asignar colores únicos a cada componente para facilitar su visualización e interpretación.
- Extraer regiones individuales para su análisis posterior o para alimentar etapas posteriores de un sistema de visión artificial.

En resumen, el análisis de componentes conexos constituye un paso clave en la estructuración de la información visual, y es la base de numerosos algoritmos avanzados en visión por computador.

12.2.1. Conector 4

Los pixeles están conectados si sus bordes se tocan. Esto significa que un par de pixeles adyacentes forman parte del mismo objeto solo si ambos están dentro y están conectados a lo largo de la dirección horizontal o vertical. Véase la Fig.12.1 para el conector 4.

12.2.2. Conector 8

Los pixeles están conectados si sus bordes o esquinas se tocan. Esto significa que un par de pixeles adyacentes están dentro, forman parte del mismo objeto, independientemente de si están conectados a lo largo de la dirección horizontal, vertical, o diagonal. Véase la Fig.12.1 para el conector 4.

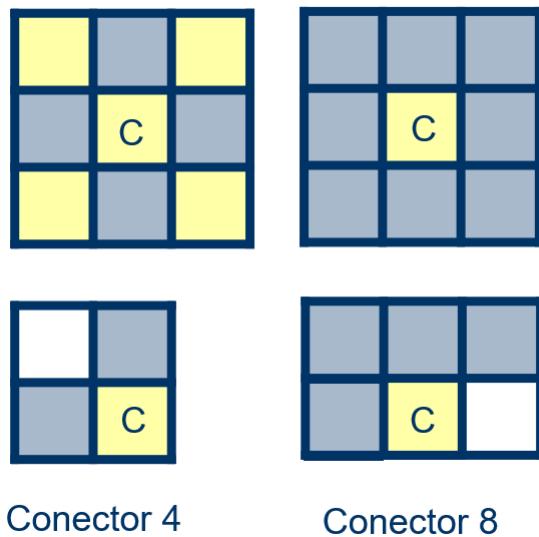


Figura 12.1: Conectores bidimensionales.

12.2.3. Proceso de Etiquetado

Este permite identificar y numerar de forma única cada región u objeto presente en una imagen binaria. El procedimiento se realiza comúnmente mediante un algoritmo de **dos pasadas**, que garantiza tanto la identificación precisa de regiones como la unificación de etiquetas equivalentes [13, 46].

Primera pasada: asignación inicial y registro de equivalencias

1. Se recorre la imagen **de izquierda a derecha y de arriba hacia abajo**, evaluando cada píxel uno a uno.
2. Para cada píxel que forma parte de un objeto (es decir, diferente del fondo), se revisan sus **vecinos ya procesados** (por ejemplo, el de arriba y el de la izquierda en conectividad 4).
3. Si **ningún vecino tiene etiqueta asignada**, se asigna una **nueva etiqueta** al píxel actual.
4. Si **uno o más vecinos tienen etiqueta**, se asigna al píxel la **etiqueta más pequeña** entre ellas.
5. Si los vecinos tienen etiquetas **diferentes**, se registra que dichas etiquetas son **equivalentes** (es decir, pertenecen al mismo componente conectado).

Este paso puede producir **etiquetas redundantes** para regiones conectadas de forma indirecta, lo cual se resuelve en la segunda pasada.

Segunda pasada: resolución de equivalencias

1. Se vuelve a recorrer la imagen en el mismo orden.
2. Para cada píxel etiquetado, se reemplaza su valor por la **etiqueta equivalente mínima**, utilizando la información de equivalencias registrada en la primera pasada.
3. Al finalizar este recorrido, todas las etiquetas redundantes se habrán **unificado correctamente**, y cada componente conectado tendrá una etiqueta única y consistente en toda su extensión.

Este comportamiento puede observarse claramente en la Figura 12.2, donde la imagen de la primera pasada presenta etiquetas múltiples dentro de una misma región, mientras que en la segunda pasada las etiquetas se han unificado para cada componente.

La figura 12.2 muestra el proceso de etiquetado para una imagen binaria de 6x6.

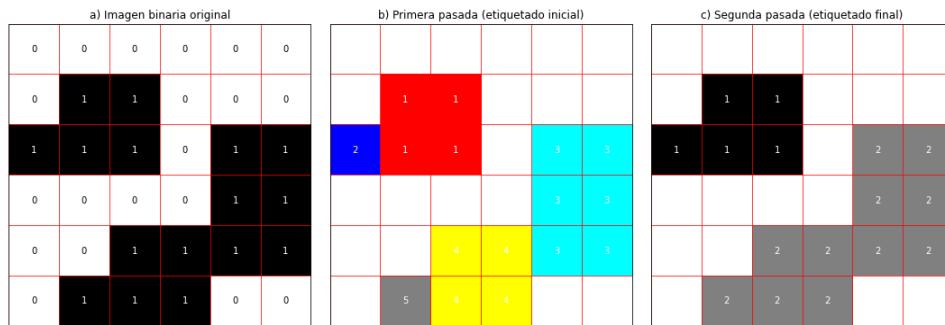


Figura 12.2: Proceso de etiquetado para una imagen binaria de 6x6.

12.2.4. Ejemplo en clase: etiquetado de regiones

En este ejemplo se muestra una imagen binaria (izquierda) compuesta por píxeles con valor 1 (en azul oscuro) y 0 (en celeste). El objetivo es etiquetar los componentes conectados utilizando conectividad 4, asignando a cada grupo de píxeles conectados una etiqueta numérica única. Es de recordar, la conectividad 4 considera como conectados a aquellos píxeles vecinos en las direcciones arriba, abajo, izquierda y derecha, es decir, no se incluyen las diagonales.

Solución

El proceso de etiquetado se realiza paso a paso, resolviendo colisiones (etiquetas equivalentes) cuando se detecta que dos regiones inicialmente etiquetadas como distintas en realidad están conectadas. Esto puede observarse, por ejemplo, cuando dos etiquetas deben unirse porque comparten un píxel común a través de conectividad 4. Véase la Figura 12.3 y siga los pasos:

- **Primera fila:** Se encuentra el primer píxel con valor binario 1, el cual no tiene vecinos etiquetados dentro del alcance del conector 4 (es decir, arriba o izquierda). Por lo tanto, se le asigna la **etiqueta 1**.
- **Segunda fila:** Se detecta otro píxel con valor 1 que tampoco tiene vecinos etiquetados; se le asigna una **nueva etiqueta (2)**. Luego, se encuentra otro píxel con valor 1 que sí tiene vecinos etiquetados (con etiquetas 1 y 2). En este caso, se asigna la **etiqueta menor (1)** y se registra una **colisión** entre las etiquetas 1 y 2 (ver círculo rojo en la figura).
- **Filas siguientes:** Se continúa el proceso, asignando nuevas etiquetas a los píxeles con valor 1 que no tienen

vecinos etiquetados, y propagando etiquetas existentes a los que sí los tienen. Cada vez que se detectan múltiples etiquetas en la vecindad, se asigna la menor y se registra la **colisión** correspondiente.

- **Cuarta fila:** Se identifica otra colisión entre etiquetas (por ejemplo, entre las etiquetas 4 y 5), y nuevamente se selecciona la menor. Esta situación también está marcada con un círculo en la figura.
- **Resolución de colisiones:** Una vez finalizado el recorrido de la imagen, se unifican las etiquetas equivalentes. Las colisiones detectadas, en este caso [1, 2] y [3, 4, 5, 6], se resuelven asignando una **única etiqueta** a cada conjunto de etiquetas equivalentes.
- **Resultado final:** La imagen etiquetada (última imagen, esquina inferior derecha) muestra una matriz con valores enteros, donde cada número representa un **componente conectado distinto**, según la conectividad 4.

12.2.5. Ejemplo Python: Etiquetado de regiones

En esta sección se implementa el **etiquetado de componentes conectados** de la imagen binaria de tools.jpg mediante el clásico **algoritmo de dos pasadas**, considerando conectividad 4. La imagen procesada que se utiliza como punto de partida (almacenada en la variable `Ic`) ya fue obtenida previamente mediante operaciones de binarización y cierre morfológico en la [Sección 11.8.2](#), por lo que aquí se explica la fase de **etiquetado** y la posterior **selección de una región** de interés.

El objetivo es asignar una *etiqueta entera a cada conjunto de píxeles conectados* (según la conectividad definida), visualizar los componentes identificados en falso color, y finalmente extraer uno de ellos para mostrarlo como una máscara sobre la imagen original en color.

Solución

```
1 #Etiquetar la Imagen
2
3 # Etiquetar componentes conectados
4 L, N = bwlabel(Ic, EE=4)
5
6 Ifalso=label2rgb(L,colormap_name='jet')
7
8 # Seleccione la etiqueta
9 Herramienta=4
10
11 # Etiqueta seleccionada
12 Selected=L==Herramienta
13
14 # Asegurarse de que la imagen en color tenga la misma dimensión que la máscara
15 # Repetir la máscara para cada canal de color
16 Selected_rgb = np.dstack([Selected] * 3)
17
18 # Aplicar la máscara de color
19 Mask=Icolor*Selected_rgb
20
21 # Mostrar la imagen tras cierre morfológico y la imagen etiquetada en un subplot
22 plt.figure(figsize=(10, 5))
23
24 # Mostrar la imagen etiquetada
25 plt.subplot(1, 3, 1)
26 plt.imshow(L, cmap='gray')
27 plt.title(f"Imagen etiquetada con {N} \n componentes conectados")
28 plt.axis('off')
29
30 plt.subplot(1, 3, 2)
31 plt.imshow(Ifalso)
32 plt.title("Falso Color RGB")
33 plt.axis('off')
34
35 plt.subplot(1, 3, 3)
36 plt.imshow(Mask)
```

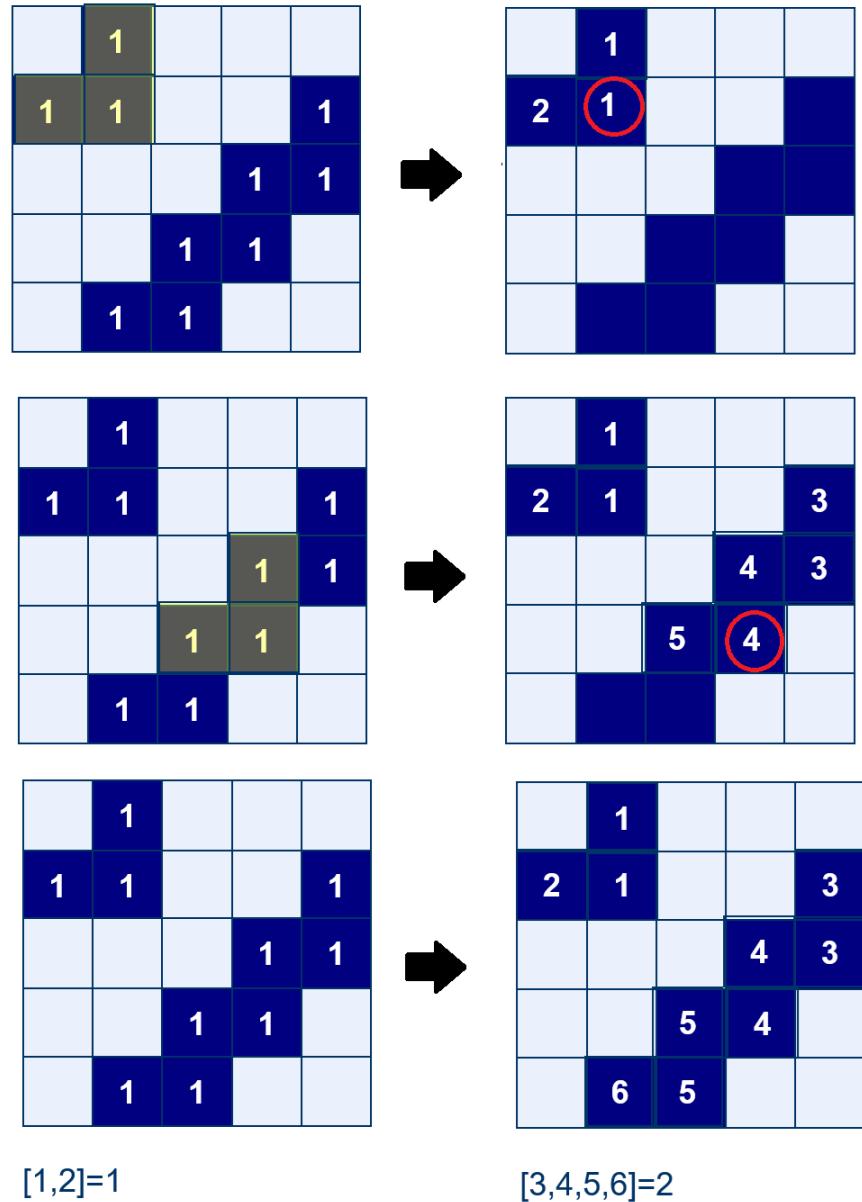


Figura 12.3: Ejemplo del proceso de etiquetado de componentes conectados en una imagen binaria utilizando conectividad 4. Se muestra la asignación inicial de etiquetas, la detección de colisiones y la unificación final de etiquetas equivalentes.

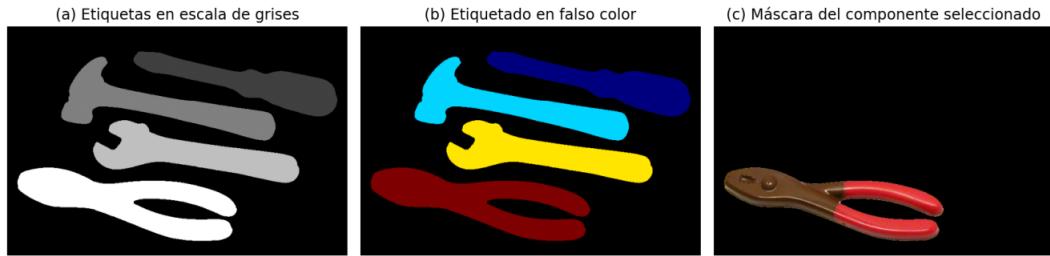


Figura 12.4: Resultados del etiquetado de componentes conectados.(a) Imagen con las etiquetas mostradas en escala de grises.(b) Visualización de los componentes conectados en falso color RGB para facilitar su identificación.(c) Máscara aplicada sobre la imagen original a color, resaltando únicamente el componente seleccionado.

```

37 plt.title("Máscara")
38 plt.axis('off')
39
40 plt.tight_layout()
41 plt.show()

```

En la Fig.12.4 se muestran tres imágenes organizadas para ilustrar el proceso de etiquetado de componentes conectados en una imagen binaria. Este procedimiento continúa el flujo iniciado en la Sección 11.8.2, donde se aplicaron operaciones morfológicas para prepararla. La imagen (a) corresponde al resultado del etiquetado aplicado sobre la imagen cerrada I_c , donde cada componente conectado ha sido asignado a una etiqueta numérica distinta, empleando conectividad 4. Los valores numéricos de las etiquetas se representan en escala de grises, siendo apenas distinguibles visualmente. En (b), se presenta la misma imagen etiquetada, pero convertida a falso color RGB utilizando un mapa de colores tipo jet, con el fin de facilitar la identificación visual de los diferentes componentes. Finalmente, la imagen (c) muestra el resultado de aplicar una máscara sobre la imagen original a color, resaltando únicamente el componente seleccionado por su etiqueta (en este caso, la etiqueta número 4). Esto permite aislar visualmente un objeto de interés. Las funciones referidas allí, se encuentran en el capítulo 13.

12.3. Funciones

Funciones principales utilizadas en el proceso de etiquetado

A continuación se describen brevemente las funciones principales empleadas en el proceso de etiquetado de componentes conectados en la imagen binaria:

1. `bwlabel(I, EE)`: Etiqueta los componentes conectados en una imagen binaria I , utilizando la conectividad definida por EE (usualmente 4 u 8). Devuelve una imagen con enteros positivos que representan las etiquetas asignadas a cada región conectada.
2. `label2rgb(L, colormap_name='jet')`: Convierte una imagen de etiquetas L en una imagen RGB para visualización, asignando un color distinto a cada etiqueta. El parámetro `colormap_name` define la paleta de colores a usar (por ejemplo, '`jet`').

12.4. Preguntas

12.4.1. Preguntas sobre hechos indicados en el texto

1. ¿Qué tipo de imágenes se utilizan principalmente para aplicar el análisis de componentes conexos?
2. ¿Cuáles son las dos formas de conectividad descritas para definir la vecindad entre píxeles en una imagen binaria?
3. ¿En qué consiste la primera pasada del algoritmo de etiquetado de componentes conectados?

4. ¿Qué operación se realiza en la segunda pasada del proceso de etiquetado?
5. ¿Qué es una colisión en el algoritmo de etiquetado presentado?

12.4.2. Preguntas de análisis y comprensión con base en el texto

1. ¿Por qué es necesario un algoritmo de dos pasadas en el proceso de etiquetado de componentes conectados, y qué soluciona su uso?
2. Explique cómo el uso de conectividad 4 en lugar de conectividad 8 puede afectar la cantidad y forma de los componentes detectados.
3. Analice el papel del registro de equivalencias durante la primera pasada. ¿Qué pasaría si no se almacenaran estas equivalencias?
4. ¿Qué ventajas ofrece la visualización en falso color de los componentes conectados frente a una imagen etiquetada en escala de grises?
5. El texto menciona que el etiquetado permite individualizar objetos para análisis posteriores. ¿Para qué podría usarse posteriormente?

12.4.3. Ejercicio numérico

Ejercicio: Comparación entre conectividad 4 y conectividad 8

Instrucciones: Aplique el algoritmo de etiquetado de componentes conectados a la siguiente imagen binaria de 7×7 , primero utilizando **conectividad 4** y luego **conectividad 8**. Realice las dos pasadas del algoritmo en cada caso y determine el número total de componentes conectados identificados en cada situación.

Imagen binaria:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Preguntas guía:

1. Realice el etiquetado completo utilizando conectividad 4. ¿Cuántos componentes conectados se detectan?
2. Repita el proceso utilizando conectividad 8. ¿Cuántos componentes conectados se detectan ahora?
3. ¿Qué diferencias observa entre los resultados obtenidos con conectividad 4 y conectividad 8?
4. ¿Qué regiones que estaban separadas en conectividad 4 se fusionaron en conectividad 8? Justifique su respuesta.
5. ¿Qué implicaciones podrían tener estas diferencias en aplicaciones como el conteo de objetos o el análisis de formas?

Capítulo 13

Funciones de Python

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib as mpl
4 import random
5
6 #-----
7 def imhist(r, ax=None, ver=True):
8     """
9         Calcula y muestra el histograma de una imagen en escala
10            de grises.
11
12    Parámetros:
13        r (numpy.ndarray): Imagen en escala de grises representada
14            como un arreglo 2D de numpy.
15        ax (matplotlib.axes.Axes, opcional): Eje de Matplotlib donde
16            se dibujará el histograma. Si no se proporciona, se usará
17            el eje actual.
18        ver (bool, opcional): Si es True, se muestra el histograma
19            usando Matplotlib.
20            Si es False, se devuelve el histograma como un arreglo.
21
22    Retorna:
23        numpy.ndarray: Si 'ver' es False, retorna un arreglo 1D
24            de numpy con el histograma de la imagen. Si 'ver'
25            es True, no retorna nada.
26
27    Ejemplo de uso:
28    >>> import numpy as np
29    >>> import matplotlib.pyplot as plt
30    >>> from ip_functions import imhist
31    >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
32    >>> imhist(img)
33    """
34    h = np.zeros([256, 1])
35    i, j = r.shape
36    for x in range(i):
37        for y in range(j):
38            h[r[x, y]] += 1
39
40    if ver:
41        x = range(256)
42        hbar = np.reshape(h, 256)
43        if ax is None:
44            ax = plt.gca()
45        ax.bar(x, hbar)
46        norm = mpl.colors.Normalize(vmin=0, vmax=255)
47        escala = plt.cm.ScalarMappable(cmap='gray', norm=norm)
48        escala.set_array([])
```

```

49         plt.colorbar(escala, ax=ax, orientation="horizontal", ticks=[0, 50, 100, 150, 200, 255])
50         ax.set_xlabel("Intensidades")
51         ax.set_ylabel("Frecuencia")
52         ax.set_title("Histograma")
53         ax.set_xlim(0, 255)
54         ax.set_ylim(0, npamax(h) * 0.3)
55         ax.grid(True)
56     else:
57         return h
58
59 #-----
60 def stretchlim(I, Tol=0.01):
61     """
62     Calcula los límites de estiramiento de contraste para
63     una imagen en escala de grises.
64
65     Parámetros:
66     I (numpy.ndarray): Imagen en escala de grises representada
67         como un arreglo 2D de numpy.
68     Tol (float, opcional): Tolerancia para el cálculo de los
69         límites. Debe estar en el rango [0, 1]. El valor por
70         defecto es 0.01.
71
72     Retorna:
73     tuple: Una tupla (Em, EM) donde Em es el límite inferior
74         y EM es el límite superior, ambos normalizados en
75         el rango [0, 1].
76
77     Ejemplo de uso:
78     >>> import numpy as np
79     >>> from ip_functions import stretchlim
80     >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
81     >>> Em, EM = stretchlim(img)
82     >>> print(Em, EM)
83     """
84     Em=0
85     EM=255
86     h=imhist(I,None,False)
87     i,j=np.shape(I)
88     ha=np.zeros([256,1])
89     hp=np.zeros([256,1])
90     L=256
91     for k in range(L):
92         ha[k]=np.sum(h[0:k+1])
93         hp[k]=ha[k]/(i*j)
94         if hp[k]<=Tol:
95             Em=k
96         if hp[k]<=1-Tol:
97             EM=k
98     return Em/255,EM/255
99 #-----
100 def imadjust(I,E,S=(0,1),n=1):
101     """
102     Ajusta la intensidad de una imagen en escala de grises
103     según los límites especificados.
104
105     Parámetros:
106     I (numpy.ndarray): Imagen en escala de grises representada
107         como un arreglo 2D de numpy.
108     E (tuple): Tupla (Em, EM) que representa los límites de
109         entrada normalizados en el rango [0, 1].
110     S (tuple, opcional): Tupla (Sm, SM) que representa los
111         límites de salida normalizados en el rango [0, 1].
112         El valor por defecto es (0, 1).
113     n (int, opcional): Exponente para el ajuste de intensidad.
114         El valor por defecto es 1.
115
116     Retorna:
117     numpy.ndarray: Imagen ajustada con las nuevas intensidades.
118
119     Ejemplo de uso:
```

```

120     >>> import numpy as np
121     >>> from ip_functions import imadjust
122     >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
123     >>> Em, EM = 0.2, 0.8
124     >>> adjusted_img = imadjust(img, (Em, EM))
125     >>> print(adjusted_img)
126     """
127     Em=E[0]*255
128     EM=E[1]*255
129     Sm=S[0]*255
130     SM=S[1]*255
131     I=np.float16(I)
132     Is=((SM-Sm) / (EM-Em)**n) * (np.absolute(I-Em))**n+Sm
133     #Ajusta el overflow de los valores
134     Is[np.where(Is>255)] = 255
135     return np.uint8(Is)
136
137 #-----
138 def histeq(I, hR=None):
139     """
140     Realiza la ecualización del histograma de una imagen en
141     escala de grises.
142
143     Parámetros:
144     I (numpy.ndarray): Imagen en escala de grises representada
145         como un arreglo 2D de numpy.
146     hR (numpy.ndarray, opcional): Histograma de referencia.
147         Si se proporciona, la ecualización se realizará en
148         función de este histograma. Si no se proporciona,
149         se realizará una ecualización normal.
150
151     Retorna:
152     numpy.ndarray: Imagen con el histograma ecualizado.
153
154     Ejemplo de uso:
155     >>> import numpy as np
156     >>> from ip_functions import histeq
157     >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
158     >>> eq_img = histeq(img)
159     >>> print(eq_img)
160     """
161
162     # Determinar el histograma de la imagen de entrada I
163     hI= imhist(I,None,False)
164
165     # Calcula la CDF de la imagen de entrada I
166     cdfI = np.cumsum(hI) / np.sum(hI)
167
168     if hR is None:
169         # Se realiza la ecualización normal
170         LUT = np.uint8(255 * cdfI)
171     else:
172         # Se hace la especificación del histograma
173         # Calcula la CDF de la imagen de referencia
174         cdfR = np.cumsum(hR) / np.sum(hR)
175
176         # Tabla de búsqueda (LUT) por proximidad
177         LUT = np.zeros(256, dtype=np.uint8)
178         for idx in range(256):
179             minIndex = np.argmin(np.abs(cdfR - cdfI[idx]))
180             LUT[idx] = minIndex # Se indexa desde 0
181
182     # Aplica la LUT a toda la imagen de entrada usando indexación directa
183     S = LUT[I]
184
185     return S
186
187 #-----
188 def imsplit(I):
189     """
190     Divide una imagen en sus canales RGB.

```

```

191
192     Parámetros:
193     I (numpy.ndarray): Imagen representada como un arreglo 3D
194         de numpy, donde la tercera dimensión corresponde a los
195         canales de color (RGB).
196
197     Retorna:
198     tuple: Una tupla (r, g, b) donde r, g y b son arreglos 2D
199         de numpy que representan los canales rojo, verde
200         y azul de la imagen, respectivamente.
201
202     Ejemplo de uso:
203     >>> import numpy as np
204     >>> from ip_functions import imsplit
205     >>> img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
206     >>> r, g, b = imsplit(img)
207     >>> print(r.shape, g.shape, b.shape)
208         (100, 100) (100, 100) (100, 100)
209     """
210     r=np.array(I[:, :, 0])
211     g=np.array(I[:, :, 1])
212     b=np.array(I[:, :, 2])
213     return r,g,b
214
215 #-----
216 def rgb2gray(RGB):
217     """
218     Convierte una imagen RGB a escala de grises utilizando
219     la fórmula de luminancia.
220
221     Parámetros:
222     RGB (numpy.ndarray): Imagen representada como un arreglo
223         3D de numpy, donde la tercera dimensión corresponde
224         a los canales de color (RGB).
225
226     Retorna:
227     numpy.ndarray: Imagen en escala de grises representada
228         como un arreglo 2D de numpy.
229
230     Ejemplo de uso:
231     >>> import numpy as np
232     >>> from ip_functions import rgb2gray
233     >>> img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
234     >>> gray_img = rgb2gray(img)
235     >>> print(gray_img.shape)
236         (100, 100)
237     """
238     r,g,b=imsplit(RGB)
239     gris=np.uint8(0.299*np.double(r)+0.587*np.double(g)+0.114*np.double(b))
240     return gris
241
242 #-----
243 def non_overflowing_sum(a,b):
244     """
245     Realiza la suma de dos arreglos de numpy sin desbordamiento,
246     limitando los valores resultantes al rango [0, 255].
247
248     Parámetros:
249     a (numpy.ndarray): Primer arreglo de entrada.
250     b (numpy.ndarray): Segundo arreglo de entrada.
251
252     Retorna:
253     numpy.ndarray: Arreglo resultante de la suma de `a` y `b`,
254         con valores limitados al rango [0, 255].
255
256     Ejemplo de uso:
257     >>> import numpy as np
258     >>> from ip_functions import non_overflowing_sum
259     >>> a = np.array([100, 150, 200], dtype=np.uint8)
260     >>> b = np.array([100, 150, 200], dtype=np.uint8)
261     >>> result = non_overflowing_sum(a, b)

```

```

262     >>> print(result)
263     [200 255 255]
264     """
265     c = np.uint16(a)+b
266     c[np.where(c>255)] = 255
267     c[np.where(c<0)] = 0
268     return np.uint8(c)
269
270 #-----
271 def graythresh(I):
272     """
273     Calcula un umbral global para convertir una imagen en
274     escala de grises a una imagen binaria utilizando el
275     método de Otsu.
276
277     Parámetros:
278     I (numpy.ndarray): Imagen en escala de grises representada
279         como un arreglo 2D de numpy.
280
281     Retorna:
282     float: Umbral calculado en el rango [0, 1].
283
284     Ejemplo de uso:
285     >>> import numpy as np
286     >>> from ip_functions import graythresh
287     >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
288     >>> threshold = graythresh(img)
289     >>> print(threshold)
290     """
291     h=imhist(I,None,False)
292     return otsuthresh(h)
293
294 #-----
295 def otsuthresh(h):
296     """
297     Calcula el umbral óptimo de Otsu para una distribución
298     de intensidades dada.
299
300     Parámetros:
301     h (numpy.ndarray): Histograma de intensidades de la imagen.
302
303     Retorna:
304     float: Umbral de Otsu normalizado en el rango [0, 1].
305
306     Ejemplo de uso:
307     >>> import numpy as np
308     >>> from ip_functions import otsuthresh
309     >>> hist = np.random.randint(0, 100, 256)
310     >>> threshold = otsuthresh(hist)
311     >>> print(threshold)
312     """
313     lh=len(h)
314     tam=np.sum(h)
315     maxV=0
316
317     for T in np.arange(0,lh,1):
318         Wb=np.sum(h[0:T])/tam
319         Acub=np.sum(h[0:T])
320         Acuf=np.sum(h[T+1:lh])
321         if Acub==0:
322             Ub=0
323         else:
324             Ub=np.dot(np.arange(0,T,1),h[0:T])/Acub
325         if Acuf==0:
326             Uf=0
327         else:
328             Uf=np.dot(np.arange(T+1,lh,1),h[T+1:lh])/Acuf
329         Wf=1-Wb
330         BCV=Wb*Wf*(Ub-Uf)**2;
331
332         if BCV>=maxV:

```

```

333             maxV=BCV
334             umbral=(T+1)/255
335     return umbral
336
337 #-----
338 def im2bw(I, threshold):
339     """
340     Convierte una imagen en escala de grises a una imagen binaria
341     utilizando un umbral especificado o calculado automáticamente.
342
343     Parámetros:
344     I (numpy.ndarray): Imagen en escala de grises representada como
345         un arreglo 2D de numpy.
346     threshold (float, opcional): Umbral para la binarización.
347         Si no se proporciona, se calculará automáticamente
348         utilizando el método de Otsu.
349
350     Retorna:
351     numpy.ndarray: Imagen binaria representada como un arreglo 2D
352         de numpy con valores 0 y 1.
353
354     Ejemplo de uso:
355     >>> import numpy as np
356     >>> from ip_functions import im2bw
357     >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
358     >>> binary_img = im2bw(img)
359     >>> print(binary_img)
360     """
361
362     # Convierte la imagen I a binaria usando el umbral especificado
363     return (I >= threshold * 255).astype(np.uint8)
364
365 #-----
366 def adaptthresh(I, P=None, V=None):
367     """
368     Calcula un umbral adaptativo para cada píxel de una imagen
369     en escala de grises utilizando el método de la media ponderada
370     de los vecinos.
371
372     Parámetros:
373     I (numpy.ndarray): Imagen en escala de grises representada
374         como un arreglo 2D de numpy.
375     P (float, opcional): Sensibilidad del umbral adaptativo.
376         Debe estar en el rango [0, 1]. El valor por defecto es 0.5.
377     V (tuple, opcional): Tamaño de la ventana de vecindario para el
378         cálculo del umbral adaptativo. El valor por defecto es 2
379         veces el tamaño de la imagen dividido por 16.
380
381     Retorna:
382     numpy.ndarray: Imagen binaria representada como un arreglo 2D de
383         numpy con valores 0 y 1.
384
385     Ejemplo de uso:
386     >>> import numpy as np
387     >>> from ip_functions import adaptthresh
388     >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
389     >>> binary_img = adaptthresh(img)
390     >>> print(binary_img)
391     """
392
393     # Valores por defecto
394     S = 0.5 # Sensibilidad por defecto
395     W = 2 * np.floor(np.array(I.shape) / 16).astype(int) + 1 # Tamaño de ventana por defecto
396
397     # Si no se proporciona V, usar la fórmula por defecto
398     if V is None:
399         V = W
400
401     # Si no se proporciona P, usar el valor por defecto
402     if P is None:
403         P = S
404
405     # Parámetros de la ventana

```

```

404     Tx, Ty = V
405     Finix = (Tx + 1) // 2
406     Ciniy = (Ty + 1) // 2
407     Ffinx = Finix - 1
408     Cfny = Ciniy - 1
409
410     # Preparar la imagen con padarray replicado
411     Io = I.copy()
412     I_padded = np.pad(Io, ((Ffinx, Ffinx), (Cfny, Cfny)), mode='edge')
413     F, C = Io.shape
414     T = np.zeros((F, C))
415
416     # Calcular el umbral local
417     for i in range(Ffinx, F - Ffinx):
418         for j in range(Ciniy, C - Cfny):
419             # Definir límites del vecindario
420             start_i = i - Ffinx
421             end_i = i + Ffinx + 1
422             start_j = j - Cfny
423             end_j = j + Cfny + 1
424
425             # Extraer el vecindario y calcular el umbral local
426             W = I_padded[start_i:end_i, start_j:end_j]
427             T[i, j] = np.mean(W) * (1 - P) # Aplicar la sensibilidad P al cálculo del umbral
428
429
430     return T/255
431
432 #-----
433 def imbinarize(I, *args):
434     """
435     Convierte una imagen en escala de grises a una imagen
436     binaria utilizando un umbral especificado o calculado
437     automáticamente.
438
439     Parámetros:
440     I (numpy.ndarray): Imagen en escala de grises representada
441         como un arreglo 2D de numpy.
442     *args: Argumentos adicionales para especificar el umbral o
443         el método de binarización. Puede ser un umbral específico,
444         el modo de binarización ('global' o 'adaptive') y argumentos
445         adicionales para el modo adaptativo.
446
447     Retorna:
448     numpy.ndarray: Imagen binaria representada como un arreglo 2D de numpy
449         con valores 0 y 1.
450
451     Ejemplo de uso:
452     >>> import numpy as np
453     >>> from ip_functions import imbinarize
454     >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
455     >>> binary_img = imbinarize(img, 128) # Usar un umbral específico
456     >>> print(binary_img)
457     >>> binary_img_auto = imbinarize(img) # Usar umbral automático
458     >>> print(binary_img_auto)
459     """
460
461     # Comprobar si el primer argumento es un escalar (umbral)
462     if len(args) == 1 and isinstance(args[0], (int, float, np.ndarray)):
463         T = args[0]
464         return im2bw(I, T)
465
466     mode = args[0].lower() if len(args) > 0 and isinstance(args[0], str) else 'global'
467     kwargs = dict(zip(args[1::2], args[2::2])) # Extraer argumentos adicionales
468
469     # Convertir todas las claves de kwargs a minúsculas
470     kwargs = {k.lower(): v for k, v in kwargs.items()}
471
472     if mode == 'global':
473         # Umbral global utilizando el método de Otsu
474         Thres = graythresh(I)
475         bw = im2bw(I, Thres)

```

```

475     elif mode == 'adaptive':
476         # Umbral adaptativo utilizando el método de Bradley
477         sensitivity = kwargs.get('sensitivity', 0.5) # Usar 0.5 si no se proporciona
478         window_size = kwargs.get('windowsize', None)
479         Tadap = adaptthresh(I, P=sensitivity, V=window_size)
480         bw = im2bw(I, Tadap)
481     else:
482         raise ValueError("Modo no válido. Debe ser 'global', 'adaptive' o proporcionar el umbral T.")
483
484     return bw
485
486 #-----
487 def immse(Iref, I):
488     """
489     Calcula el error cuadrático medio (MSE) entre dos imágenes.
490
491     Parámetros:
492     I1 (numpy.ndarray): Primera imagen representada como un
493         arreglo 2D de numpy.
494     I2 (numpy.ndarray): Segunda imagen representada como un
495         arreglo 2D de numpy.
496
497     Retorna:
498     float: El error cuadrático medio entre las dos imágenes.
499
500     Ejemplo de uso:
501     >>> import numpy as np
502     >>> from ip_functions import immse
503     >>> img1 = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
504     >>> img2 = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
505     >>> mse = immse(img1, img2)
506     >>> print(mse)
507     """
508
509     fil,col=np.shape(Iref)
510     I1=np.array(Iref,dtype=float)
511     I2=np.array(I,dtype=float)
512     MSE=np.sum((I1-I2)**2)/(fil*col)
513     return MSE
514
515 #-----
516 def psnr(Iref, I):
517     """
518     Calcula el error cuadrático medio (MSE) entre dos imágenes.
519
520     Parámetros:
521     Iref (numpy.ndarray): Primera imagen representada como un
522         arreglo 2D de numpy.
523     I (numpy.ndarray): Segunda imagen representada como un
524         arreglo 2D de numpy.
525
526     Retorna:
527     float: El error cuadrático medio entre las dos imágenes.
528
529     Ejemplo de uso:
530     >>> import numpy as np
531     >>> from ip_functions import immse
532     >>> img1 = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
533     >>> img2 = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
534     >>> mse = immse(img1, img2)
535     >>> print(mse)
536
537     I2=np.array(I,dtype=float)
538     fil,col=np.shape(Iref)
539     A=immse(Iref, I)
540     psnr=10*np.log10(255**2/A)
541     Mean_noise=np.mean(I2**2)
542     snr=10*np.log10(Mean_noise/A)
543     return psnr,snr
544
545 #-----
546 def imnoise(I, tipo='gaussian', P=0.5, sigma=0.1):
547     """

```

```

546     Añade ruido a una imagen en escala de grises.
547
548     Parámetros:
549     I (numpy.ndarray): Imagen en escala de grises representada
550         como un arreglo 2D de numpy.
551     tipo (str, opcional): Tipo de ruido a añadir. Puede ser
552         'salt & pepper', 'gaussian' o 'speckle'. El valor por
553         defecto es 'gaussian'.
554     P (float, opcional): Probabilidad de ruido para el ruido de sal
555         y pimienta. El valor por defecto es 0.5.
556     sigma (float, opcional): Desviación estándar para el ruido gaussiano
557         o speckle. El valor por defecto es 0.1.
558
559     Retorna:
560     numpy.ndarray: Imagen con ruido añadido.
561
562     Ejemplo de uso:
563     >>> import numpy as np
564     >>> from ip_functions import imnoise
565     >>> img = np.random.randint(0, 256, (100, 100), dtype=np.uint8)
566     >>> noisy_img = imnoise(img, mode='gaussian', mean=0, var=0.01)
567     >>> print(noisy_img)
568     """
569     mu=0
570     fil,col=np.shape(I)
571     pnt=int(fil*col*P)
572     tipo=tipo.lower()
573     if tipo=='salt & pepper':
574         Ir=np.copy(I)
575         for i in range(pnt):
576             x=random.randint(0, fil-1)
577             y=random.randint(0, col-1)
578             Ir[x,y]=255*random.randint(0,1)
579             Ir=np.uint8(Ir)
580
581     elif tipo=='gaussian':
582         mu=P
583         s = np.random.normal(mu, sigma*255, (fil,col))
584         Ir=non_overflowing_sum(I,s)
585
586     elif tipo=='speckle':
587         sigma=P
588         s = np.random.normal(mu, sigma, (fil,col))
589         Ir=non_overflowing_sum(I,s*I)
590         Ir=np.uint8(Ir)
591     else:
592         Ir=np.copy(I)
593
594     return Ir
595
596
597 #-----
598
599 def rgb2hsv(I):
600     """
601     Convierte una imagen de color RGB a una imagen en el
602     espacio de color HSV.
603
604     Parámetros:
605     I (numpy.ndarray): Imagen representada como un arreglo
606         3D de numpy, donde la tercera dimensión corresponde
607         a los canales de color (RGB).
608
609     Retorna:
610     numpy.ndarray: Imagen en el espacio de color HSV representada
611         como un arreglo 3D de numpy, donde la tercera dimensión
612         corresponde a los canales de color (HSV).
613
614     Ejemplo de uso:
615     >>> import numpy as np
616     >>> from ip_functions import rgb2hsv

```

```

617     >>> img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
618     >>> hsv_img = rgb2hsv(img)
619     >>> print(hsv_img.shape)
620     (100, 100, 3)
621     """
622     I=I/255
623     r,g,b=imsplit(I)
624     [fil,col,pro]=np.shape(I)
625     Cmax=np.zeros((fil,col))
626     Cmin=np.zeros((fil,col))
627     d=np.zeros((fil,col))
628     H=np.zeros((fil,col))
629     S=np.zeros((fil,col))
630     V=np.zeros((fil,col))
631     for i in range(fil):
632         for j in range(col):
633             maximo=max([r[i,j],g[i,j],b[i,j]])
634             minimo=min([r[i,j],g[i,j],b[i,j]])
635             Cmax[i,j]=maximo
636             Cmin[i,j]=minimo
637             d[i,j]=maximo-minimo
638             if d[i,j]==0:
639                 H[i,j]=0
640             elif maximo==r[i,j]:
641                 H[i,j]=60*((g[i,j]-b[i,j])/d[i,j])%6
642             elif maximo==g[i,j]:
643                 H[i,j]=60*((g[i,j]-b[i,j])/d[i,j])+2
644             elif maximo==b[i,j]:
645                 H[i,j]=60*((g[i,j]-b[i,j])/d[i,j])+4
646             if maximo==0:
647                 S[i,j]=0
648             else:
649                 S[i,j]=d[i,j]/maximo
650             V[i,j]=maximo
651     H=H/360.0
652     hsv=np.dstack((H,S,V))
653     return hsv
654
655 #-----
656
657 def hsv2rgb(H):
658     """
659     Convierte una imagen en el espacio de color HSV a una
660     imagen de color RGB.
661
662     Parámetros:
663     H (numpy.ndarray): Imagen representada como un arreglo 3D de numpy,
664         donde la tercera dimensión corresponde a los canales
665         de color (HSV).
666
667     Retorna:
668     numpy.ndarray: Imagen en el espacio de color RGB representada
669         como un arreglo 3D de numpy, donde la tercera dimensión
670         corresponde a los canales de color (RGB).
671
672     Ejemplo de uso:
673     >>> import numpy as np
674     >>> from ip_functions import hsv2rgb
675     >>> hsv_img = np.random.rand(100, 100, 3) # Valores en el rango [0, 1] para H, S, V
676     >>> rgb_img = hsv2rgb(hsv_img)
677     >>> print(rgb_img.shape)
678     (100, 100, 3)
679     """
680     h,s,v=imsplit(H)
681     N,M,L=np.shape(H)
682     X=np.zeros((N,M))
683     m=np.zeros((N,M))
684     r=np.zeros((N,M))
685     g=np.zeros((N,M))
686     b=np.zeros((N,M))
687     h=h*360

```

```

688 C=np.zeros((N,M)) #s*v
689 for i in range(N):
690     for j in range(M):
691         C[i,j]=v[i,j]*s[i,j]
692         X[i,j]=C[i,j]*(1-abs((h[i,j]/60)%2-1))
693         m[i,j]=v[i,j]-C[i,j]
694         if 0<=h[i,j] and h[i,j]<60:
695             r[i,j],g[i,j],b[i,j]=C[i,j],X[i,j],0
696         elif 60<=h[i,j] and h[i,j]<120:
697             r[i,j],g[i,j],b[i,j]=X[i,j],C[i,j],0
698         elif 120<=h[i,j] and h[i,j]<180:
699             r[i,j],g[i,j],b[i,j]=0,C[i,j],X[i,j]
700         elif 180<=h[i,j] and h[i,j]<240:
701             r[i,j],g[i,j],b[i,j]=0,X[i,j],C[i,j]
702         elif 240<=h[i,j] and h[i,j]<300:
703             r[i,j],g[i,j],b[i,j]=X[i,j],0,C[i,j]
704         elif 300<=h[i,j] and h[i,j]<360:
705             r[i,j],g[i,j],b[i,j]=C[i,j],0,X[i,j]
706 R,G,B=255*(r+m),255*(g+m),255*(b+m)
707 RGB=np.dstack((R,G,B))
708 RGB=np.uint8(RGB)
709 return RGB
710
711 #-----
712 def xyz2lab(myXYZ):
713     """
714     Convierte una imagen en el espacio de color XYZ a una
715     imagen en el espacio de color CIELAB.
716
717     Parámetros:
718     myXYZ (numpy.ndarray): Imagen representada como un arreglo
719         3D de numpy, donde la tercera dimensión corresponde
720         a los canales de color (XYZ).
721
722     Retorna:
723     numpy.ndarray: Imagen en el espacio de color CIELAB representada
724         como un arreglo 3D de numpy, donde la tercera dimensión
725         corresponde a los canales de color (L, a, b).
726
727     Ejemplo de uso:
728     >>> import numpy as np
729     >>> from ip_functions import xyz2lab
730     >>> xyz_img = np.random.rand(100, 100, 3) # Valores en el rango [0, 1] para X, Y, Z
731     >>> lab_img = xyz2lab(xyz_img)
732     >>> print(lab_img.shape)
733     (100, 100, 3)
734     """
735     F, C, L = myXYZ.shape
736     if L != 3:
737         raise ValueError('xyz2lab: La imagen debe ser MxNx3')
738
739     X = myXYZ[:, :, 0]
740     Y = myXYZ[:, :, 1]
741     Z = myXYZ[:, :, 2]
742
743     CIELAB = np.zeros((F, C, L))
744     CIEL = np.zeros((F, C))
745     CIEa = np.zeros((F, C))
746     CIEb = np.zeros((F, C))
747
748     # D65
749     M = np.array([
750         [0.4124564, 0.3575761, 0.1804375],
751         [0.2126729, 0.7151522, 0.0721750],
752         [0.0193339, 0.1191920, 0.9503041]
753     ])
754
755     Reference = M @ np.array([100, 100, 100]) # Respecto a 100 en D65
756     ReferenceX, ReferenceY, ReferenceZ = Reference / 100
757
758     var_X = X / ReferenceX

```

```

759     var_Y = Y / ReferenceY
760     var_Z = Z / ReferenceZ
761
762     def evaluar(x):
763         return np.where(x > 0.008856, x** (1/3), 7.787 * x + 16/116)
764
765     var_X = evaluar(var_X)
766     var_Y = evaluar(var_Y)
767     var_Z = evaluar(var_Z)
768
769     CIEL = 116 * var_Y - 16
770     CIEa = 500 * (var_X - var_Y)
771     CIEb = 200 * (var_Y - var_Z)
772
773     CIELAB[:, :, 0] = CIEL
774     CIELAB[:, :, 1] = CIEa
775     CIELAB[:, :, 2] = CIEb
776
777     return CIELAB.astype(np.float64)
778
779
780 #-----
781 def lab2xyz(lab):
782     """
783     Convierte una imagen en el espacio de color CIELAB a una
784     imagen en el espacio de color XYZ.
785
786     Parámetros:
787     lab (numpy.ndarray): Imagen representada como un arreglo 3D
788         de numpy, donde la tercera dimensión corresponde a
789         los canales de color (L*, a*, b*).
790
791     Retorna:
792     numpy.ndarray: Imagen en el espacio de color XYZ representada
793     como un arreglo 3D de numpy, donde la tercera dimensión corresponde
794     a los canales de color (X, Y, Z).
795
796     Ejemplo de uso:
797     >>> import numpy as np
798     >>> from ip_functions import lab2xyz
799     >>> lab_img = np.random.rand(100, 100, 3) * [100, 255, 255] # Valores en el rango [0, 100] para L y [-128, 127] pa
800     >>> xyz_img = lab2xyz(lab_img)
801     >>> print(xyz_img.shape)
802     (100, 100, 3)
803     """
804     # Asegurarse de que el último eje tenga tamaño 3
805     if lab.shape[-1] != 3:
806         raise ValueError("El último eje debe tener tamaño 3 (L*, a*, b*)")
807
808     # Referencia D65 a 2 grados
809     Reference = np.array([0.950456, 1.000000, 1.088754])
810
811     def evaluar(x):
812         return np.where(x > 0.008856, x ** 3, (x - 16 / 116) / 7.787)
813
814     L, a, b = lab[:, :, 0], lab[:, :, 1], lab[:, :, 2]
815
816     nL = (L + 16) / 116
817     na = a / 500
818     nb = b / 200
819
820     var_X = evaluar(nL + na)
821     var_Y = evaluar(nL)
822     var_Z = evaluar(nL - nb)
823
824     XYZ = np.zeros_like(lab)
825     XYZ[:, :, 0] = Reference[0] * var_X
826     XYZ[:, :, 1] = Reference[1] * var_Y
827     XYZ[:, :, 2] = Reference[2] * var_Z
828
829     return XYZ

```

```

830
831 #-----
832 def xyz2rgb(XYZ):
833     """
834     Convierte una imagen en el espacio de color XYZ a una imagen
835     en el espacio de color RGB.
836
837     Parámetros:
838     XYZ (numpy.ndarray): Imagen representada como un arreglo 3D
839         de numpy, donde la tercera dimensión corresponde a
840         los canales de color (X, Y, Z).
841
842     Retorna:
843     numpy.ndarray: Imagen en el espacio de color RGB representada
844         como un arreglo 3D de numpy, donde la tercera dimensión
845         corresponde a los canales de color (R, G, B).
846
847     Ejemplo de uso:
848     >>> import numpy as np
849     >>> from ip_functions import xyz2rgb
850     >>> xyz_img = np.random.rand(100, 100, 3) # Valores en el rango [0, 1] para X, Y, Z
851     >>> rgb_img = xyz2rgb(xyz_img)
852     >>> print(rgb_img.shape)
853     (100, 100, 3)
854     """
855     # Asegurarse de que el último eje tenga tamaño 3
856     if XYZ.shape[-1] != 3:
857         raise ValueError("El último eje debe tener tamaño 3 (X, Y, Z)")
858
859     def evaluar(x):
860         return np.where(x > 0.0031308,
861                         1.055 * np.power(np.maximum(x, 0), 1 / 2.4) - 0.055,
862                         12.92 * x)
863
864     M = np.array([
865         [ 3.2404542, -1.5371385, -0.4985314],
866         [-0.9692660,  1.8760108,  0.0415560],
867         [ 0.0556434, -0.2040259,  1.0572252]
868     ])
869
870     var_RGB = np.dot(XYZ, M.T)
871     var_RGB = np.clip(var_RGB, 0, None) # Asegura que no haya valores negativos
872
873     RGB = evaluar(var_RGB)
874     RGB = (RGB * 255).astype(np.uint8)
875
876     return RGB
877
878 #-----
879 def rgb2xyz(RGB):
880     """
881     Convierte una imagen en el espacio de color RGB a una imagen
882     en el espacio de color XYZ.
883
884     Parámetros:
885     RGB (numpy.ndarray): Imagen representada como un arreglo 3D
886         de numpy, donde la tercera dimensión corresponde a
887         los canales de color (RGB).
888
889     Retorna:
890     numpy.ndarray: Imagen en el espacio de color XYZ representada
891         como un arreglo 3D de numpy, donde la tercera dimensión
892         corresponde a los canales de color (X, Y, Z).
893
894     Ejemplo de uso:
895     >>> import numpy as np
896     >>> from ip_functions import rgb2xyz
897     >>> rgb_img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
898     >>> xyz_img = rgb2xyz(rgb_img)
899     >>> print(xyz_img.shape)
900     (100, 100, 3)

```

```

901 """
902 F, C, L = RGB.shape
903 if L != 3:
904     raise ValueError('rgb2xyz: La imagen debe ser MxNx3')
905
906 if RGB.dtype == np.uint8:
907     RGB = RGB.astype(np.float64)
908     div = 255
909 else:
910     div = 1
911
912 sR = RGB[:, :, 0]
913 sG = RGB[:, :, 1]
914 sB = RGB[:, :, 2]
915
916 var_R = sR / div
917 var_G = sG / div
918 var_B = sB / div
919
920 def evaluar(x):
921     return np.where(x > 0.04045, ((x + 0.055) / 1.055) ** 2.4, x / 12.92)
922
923 var_R = evaluar(var_R)
924 var_G = evaluar(var_G)
925 var_B = evaluar(var_B)
926
927 M = np.array([
928     [0.4124564, 0.3575761, 0.1804375],
929     [0.2126729, 0.7151522, 0.0721750],
930     [0.0193339, 0.1191920, 0.9503041]
931 ])
932
933 XYZ = np.zeros((F, C, L))
934
935 for i in range(F):
936     for j in range(C):
937         var_RGB = M @ np.array([var_R[i, j], var_G[i, j], var_B[i, j]])
938         XYZ[i, j, :] = var_RGB
939
940 return XYZ
941
942 #-----
943 def rgb2lab(RGB):
944 """
945 Convierte una imagen en el espacio de color RGB a una imagen
946 en el espacio de color CIELAB.
947
948 Parámetros:
949 RGB (numpy.ndarray): Imagen representada como un arreglo 3D
950     de numpy, donde la tercera dimensión corresponde a
951     los canales de color (RGB).
952
953 Retorna:
954 numpy.ndarray: Imagen en el espacio de color CIELAB representada
955     como un arreglo 3D de numpy, donde la tercera dimensión
956     corresponde a los canales de color (L, a, b).
957
958 Ejemplo de uso:
959 >>> import numpy as np
960 >>> from ip_functions import rgb2lab
961 >>> rgb_img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
962 >>> lab_img = rgb2lab(rgb_img)
963 >>> print(lab_img.shape)
964 (100, 100, 3)
965 """
966 F, C, L = np.shape(RGB)
967 if L != 3:
968     raise ValueError('rgb2lab: La imagen debe ser MxNx3')
969
970 xyz = rgb2xyz(RGB)
971 CIEL = xyz2lab(xyz)

```

```

972     return CIEL
973
974
975 #-----
976 def lab2rgb(LAB):
977     """
978     Convierte una imagen en el espacio de color CIELAB a una imagen
979     en el espacio de color RGB.
980
981     Parámetros:
982     LAB (numpy.ndarray): Imagen representada como un arreglo 3D de
983         numpy, donde la tercera dimensión corresponde a los
984         canales de color (L, a, b).
985
986     Retorna:
987     numpy.ndarray: Imagen en el espacio de color RGB representada
988         como un arreglo 3D de numpy, donde la tercera dimensión
989         corresponde a los canales de color (R, G, B).
990
991     Ejemplo de uso:
992     >>> import numpy as np
993     >>> from ip_functions import lab2rgb
994     >>> lab_img = np.random.rand(100, 100, 3) * [100, 255, 255] # Valores en el rango [0, 100] para L y [-
995     >>> rgb_img = lab2rgb(lab_img)
996     >>> print(rgb_img.shape)
997     (100, 100, 3)
998     """
999     # Verificar si la entrada es una matriz MxNx3
1000    F, C, L = np.shape(LAB)
1001    if L != 3:
1002        raise ValueError('lab2rgb: La imagen debe ser MxNx3')
1003
1004    # Paso 1: Convertir de LAB a XYZ
1005    xyz = lab2xyz(LAB)
1006
1007    # Paso 2: Convertir de XYZ a RGB
1008    RGB = xyz2rgb(xyz)
1009
1010    return RGB
1011
1012 #-----
1013 def imrotate(I, grados):
1014     """
1015     Rota una imagen en escala de grises o en color por un
1016     ángulo especificado.
1017
1018     Parámetros:
1019     I (numpy.ndarray): Imagen representada como un arreglo 2D
1020         (escala de grises) o 3D (color) de numpy.
1021     grados (float): Ángulo de rotación en grados.
1022
1023     Retorna:
1024     numpy.ndarray: Imagen rotada.
1025
1026     Ejemplo de uso:
1027     >>> import numpy as np
1028     >>> from ip_functions import imrotate
1029     >>> img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
1030     >>> rotated_img = imrotate(img, 45)
1031     >>> print(rotated_img.shape)
1032     """
1033     # Convertir los grados a radianes
1034     theta = -grados * np.pi / 180
1035
1036     # Obtener las dimensiones de la imagen original
1037     M, N, C = I.shape # C es el número de canales (3 para RGB)
1038
1039     # Centro de la imagen original
1040     pc = np.array([N, M, 1]) / 2
1041
1042     # Matriz de rotación inversa

```

```

043     R = np.array([
044         [np.cos(theta), -np.sin(theta), 0],
045         [np.sin(theta), np.cos(theta), 0],
046         [0, 0, 1]
047     ])
048
049     R_inv = np.linalg.inv(R)
050
051     # Calcular las nuevas dimensiones de la imagen rotada
052     D = np.abs(R)
053     z = np.array([N, M, 1])
054     zp = np.dot(D, z) # Nuevas dimensiones sin el término homogéneo
055
056     # Dimensiones de la imagen rotada
057     Np = int(np.ceil(zp[0])) # Nueva anchura
058     Mp = int(np.ceil(zp[1])) # Nueva altura
059
060     # Centro de la imagen rotada
061     pc_p = np.array([Np, Mp, 1]) / 2
062
063     # Inicializar la nueva imagen rotada
064     I_rotada = np.zeros((Mp, Np, C), dtype=np.uint8)
065
066     # Ciclos for para recorrer la imagen rotada
067     for xp in range(Np):
068         for yp in range(Mp):
069
070             # Coordenadas homogéneas del píxel en la imagen rotada
071             p_p = np.array([xp, yp, 1])
072
073             # Calcular la posición relativa respecto al centro de la imagen rotada
074             p_p_rel = p_p - pc_p
075
076             # Aplicar la matriz de rotación inversa a las coordenadas relativas
077             p_rel = np.dot(R_inv, p_p_rel)
078
079             # Ajustar las coordenadas al centro de la imagen original
080             p = p_rel + pc_p
081
082             # Redondear las coordenadas al píxel más cercano
083             x = int(np.round(p[0]))
084             y = int(np.round(p[1]))
085
086             # Verificar si las coordenadas están dentro de los límites de la imagen original
087             if 0 <= x < N and 0 <= y < M:
088                 # Asignar los valores de los píxeles de la imagen original a la imagen rotada
089                 I_rotada[yp, xp, :] = I[y, x, :]
090
091     return I_rotada
092
093     #-----
094 def imcrop(I, x, y, w, h):
095     """
096     Recorta una imagen a un rectángulo especificado.
097
098     Parámetros:
099     I (numpy.ndarray): Imagen representada como un arreglo 2D
100         (escala de grises) o 3D (color) de numpy.
101     x (int): Coordenada x del punto de inicio del recorte.
102     y (int): Coordenada y del punto de inicio del recorte.
103     w (int): Ancho del rectángulo de recorte.
104     h (int): Altura del rectángulo de recorte.
105
106     Retorna:
107     numpy.ndarray: Imagen recortada.
108
109     Ejemplo de uso:
110     >>> import numpy as np
111     >>> from ip_functions import imcrop
112     >>> img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
113     >>> cropped_img = imcrop(img, (10, 10, 50, 50))

```

```

114     >>> print(cropped_img.shape)
115     (50, 50, 3)
116     """
117     if I.ndim not in [2, 3]:
118         raise ValueError("La imagen debe ser 2D (escala de grises) o 3D (color)")
119
120     height, width = I.shape[:2]
121
122     # Validar los límites del recorte
123     if x < 0 or y < 0 or x + w > width or y + h > height:
124         raise ValueError("Los parámetros de recorte están fuera de los límites de la imagen")
125
126     if I.ndim == 2: # Imagen en escala de grises
127         return I[y:y+h, x:x+w]
128     else: # Imagen en color
129         return I[y:y+h, x:x+w, :]
130
131
132 #-----
133 def imresize(I, S):
134     """
135     Cambia el tamaño de una imagen a las dimensiones especificadas.
136
137     Parámetros:
138     I (numpy.ndarray): Imagen representada como un arreglo 2D
139         (escala de grises) o 3D (color) de numpy.
140     S (int o tuple): Factor de escala o tamaño de la imagen
141         de salida en el formato (nueva_altura, nueva_anchura).
142
143     Retorna:
144     numpy.ndarray: Imagen redimensionada.
145
146     Ejemplo de uso:
147     >>> import numpy as np
148     >>> from ip_functions import imresize
149     >>> img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
150     >>> resized_img = imresize(img, (50, 50))
151     >>> print(resized_img.shape)
152     (50, 50, 3)
153     """
154
155     # Leer tamaño de la imagen original
156     N, M, L = I.shape # N: altura, M: ancho, L: número de canales
157
158     # Determinar los factores de escala
159     if np.isscalar(S):
160         Sx = Sy = S # Factor de escala en x y en y
161     else:
162         Sx, Sy = S[0], S[1] # Factor de escala en x y en y
163
164     # Construir la matriz de escalamiento
165     S_matrix = np.array([
166         [Sx, 0, 0],
167         [0, Sy, 0],
168         [0, 0, 1]
169     ])
170
171     # Calcular el nuevo tamaño de la imagen
172     z = np.array([N, M, 1]) # Dimensiones originales en formato homogéneo
173     zp = np.dot(S_matrix, z) # Nuevas dimensiones
174     Np = int(np.ceil(zp[0])) # Nueva altura
175     Mp = int(np.ceil(zp[1])) # Nuevo ancho
176
177     # Crear la nueva imagen de salida
178     Ip = np.zeros((Np, Mp, L), dtype=np.uint8)
179
180     # Método Inverso
181     for yp in range(Np):
182         for xp in range(Mp):
183             # Coordenadas homogéneas del pixel en la imagen escalada
184             pp = np.array([xp, yp, 1])

```

```

185     # Calcular la posición en la imagen original
186     S_inv = np.array([
187         [1/Sx, 0, 0],
188         [0, 1/Sy, 0],
189         [0, 0, 1]
190     ]) # Matriz inversa de escalamiento
191
192     p = np.dot(S_inv, pp) # Coordenadas originales
193
194     # Redondear las coordenadas al píxel más cercano
195     x = int(np.round(p[0]))
196     y = int(np.round(p[1]))
197
198     # Verificar si las coordenadas están dentro de los límites de la imagen original
199     if 0 <= x < M and 0 <= y < N:
200         # Asignar los valores de los píxeles de la imagen original a la imagen escalada
201         Ip[yp, xp, :] = I[y, x, :]
202
203     return Ip
204
205 #-----
206 def imtranslate(I, translation, mode='same'):
207     """
208         Traduce (desplaza) una imagen en escala de grises o en
209         color por un desplazamiento especificado.
210
211     Parámetros:
212     I (numpy.ndarray): Imagen representada como un arreglo
213         2D (escala de grises) o 3D (color) de numpy.
214     translation (tuple): Desplazamiento en píxeles en el
215         formato (tx, ty).
216     mode (str, opcional): Modo de salida de la imagen.
217         Puede ser 'same' (mantiene las mismas dimensiones)
218         o 'full' (considera el tamaño extendido). El valor
219         por defecto es 'same'.
220
221     Retorna:
222     numpy.ndarray: Imagen traducida (desplazada).
223
224     Ejemplo de uso:
225     >>> import numpy as np
226     >>> from ip_functions import imtranslate
227     >>> img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
228     >>> translated_img = imtranslate(img, (10, 20))
229     >>> print(translated_img.shape)
230     """
231
232     # Obtiene las dimensiones de la imagen original
233     N, M, L = I.shape # N: altura, M: ancho, L: canales
234
235     # Desplazamientos en x e y
236     tx, ty = translation
237
238     # Matriz de translación
239     T = np.array([
240         [1, 0, tx],
241         [0, 1, ty],
242         [0, 0, 1]
243     ])
244
245     # Si el modo es 'full', se considera el tamaño extendido
246     if mode == 'full':
247         # Cálculo de nuevas dimensiones considerando el desplazamiento
248         D = np.abs(T)
249         z = np.array([M, N, 1])
250         zp = np.dot(D, z) # Nuevas dimensiones
251         Mp = int(np.round(zp[0])) # Nuevo ancho
252         Np = int(np.round(zp[1])) # Nueva altura
253     elif mode == 'same':
254         Mp, Np = M, N # Mantiene las mismas dimensiones que la imagen original
255     else:
256         raise ValueError("El modo debe ser 'same' o 'full'")

```

```

1256     # Inicializa la nueva imagen con ceros
1257     Ip = np.zeros((Np, Mp, L), dtype=np.uint8)
1258
1259     # Recorrer cada pixel de la nueva imagen
1260     for yp in range(Np):
1261         for xp in range(Mp):
1262             # Calcula las coordenadas en la imagen original aplicando la inversa de T
1263             pp = np.array([xp, yp, 1])
1264             p = np.dot(np.linalg.inv(T), pp)
1265
1266             x = int(np.round(p[0]))
1267             y = int(np.round(p[1]))
1268
1269             # Verifica si las coordenadas están dentro de los límites de la imagen original
1270             if 0 <= x < M and 0 <= y < N:
1271                 Ip[yp, xp, :] = I[y, x, :]
1272
1273     # Convierte la imagen resultante a tipo uint8 (si es necesario)
1274     return Ip
1275
1276
1277 #-----
1278 def fitgeotrans(puntos_iniciales, puntos_finales, transformation_type='projective'):
1279     """
1280     Calcula una transformación geométrica que mapea puntos de
1281     una imagen en movimiento a puntos de una imagen fija.
1282
1283     Parámetros:
1284     puntos_iniciales (numpy.ndarray): Coordenadas de los puntos en
1285         la imagen en movimiento. Cada fila corresponde a un punto
1286         y las columnas a las coordenadas (x, y).
1287     puntos_finales (numpy.ndarray): Coordenadas de los puntos en la
1288         imagen fija. Cada fila corresponde a un punto y las columnas
1289         a las coordenadas (x, y).
1290     transformation_type (str, opcional): Tipo de transformación geométrica.
1291         Puede ser 'affine' o 'projective'. El valor por defecto es 'projective'.
1292
1293     Retorna:
1294     skimage.transform.ProjectiveTransform o skimage.transform.AffineTransform:
1295         Objeto de transformación geométrica que puede ser aplicado a una imagen.
1296
1297     Ejemplo de uso:
1298     >>> import numpy as np
1299     >>> from ip_functions import fitgeotrans
1300     >>> moving_points = np.array([[0, 0], [1, 0], [0, 1]])
1301     >>> fixed_points = np.array([[0, 0], [2, 0], [0, 2]])
1302     >>> tform = fitgeotrans(moving_points, fixed_points, 'affine')
1303     >>> print(tform)
1304     """
1305
1306     # Número de puntos
1307     n = puntos_iniciales.shape[0]
1308
1309     if transformation_type == 'affine':
1310         # Transformación afín
1311         num_params = 6
1312         A = np.zeros((2 * n, num_params))
1313         b = np.zeros(2 * n)
1314
1315         for i in range(n):
1316             x, y = puntos_iniciales[i]
1317             x_prime, y_prime = puntos_finales[i]
1318
1319             # Ecuación para x'
1320             A[2*i, :] = [x, y, 1, 0, 0, 0]
1321             b[2*i] = x_prime
1322
1323             # Ecuación para y'
1324             A[2*i+1, :] = [0, 0, 0, x, y, 1]
1325             b[2*i+1] = y_prime
1326
1327     # Resolver el sistema A * h = b

```

```

327     h = np.linalg.lstsq(A, b, rcond=None)[0]
328
329     # La matriz H tiene la forma [h1 h2 h3; h4 h5 h6; 0 0 1]
330     H = np.array([[h[0], h[1], h[2]],
331                   [h[3], h[4], h[5]],
332                   [0, 0, 1]])
333
334 elif transformation_type == 'projective':
335     # Transformación proyectiva
336     num_params = 8
337     A = np.zeros((2 * n, num_params))
338     b = np.zeros(2 * n)
339
340     for i in range(n):
341         x, y = puntos_iniciales[i]
342         x_prime, y_prime = puntos_finales[i]
343
344         # Ecuación para x'
345         A[2*i, :] = [x, y, 1, 0, 0, 0, -x_prime * x, -x_prime * y]
346         b[2*i] = x_prime
347
348         # Ecuación para y'
349         A[2*i+1, :] = [0, 0, 0, x, y, 1, -y_prime * x, -y_prime * y]
350         b[2*i+1] = y_prime
351
352     # Resolver el sistema A * h = b
353     h = np.linalg.lstsq(A, b, rcond=None)[0]
354
355     # La matriz H tiene la forma [h1 h2 h3; h4 h5 h6; h7 h8 1]
356     H = np.array([[h[0], h[1], h[2]],
357                   [h[3], h[4], h[5]],
358                   [h[6], h[7], 1]])
359
360 else:
361     raise ValueError("Tipo de transformación no soportado. Use 'affine' o 'projective'.")
362
363 return H
364
365 #-----
366 def imwarp(I, H):
367     """
368     Aplica una transformación geométrica a una imagen utilizando
369     una matriz de transformación.
370
371     Parámetros:
372     I (numpy.ndarray): Imagen representada como un arreglo 3D de numpy.
373     H (numpy.ndarray): Matriz de transformación geométrica 3x3.
374
375     Retorna:
376     numpy.ndarray: Imagen transformada.
377
378     Ejemplo de uso:
379     >>> import numpy as np
380     >>> from skimage.transform import AffineTransform
381     >>> from ip_functions import imwarp
382     >>> img = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
383     >>> tform = AffineTransform(scale=(1.5, 1.5))
384     >>> warped_img = imwarp(img, tform)
385     >>> print(warped_img.shape)
386     """
387
388     # Obtener las dimensiones de la imagen
389     N, M, L = I.shape
390
391     # Esquinas de la imagen original (en coordenadas homogéneas)
392     corners = np.array([
393         [1, 1, 1],
394         [M, 1, 1],
395         [1, N, 1],
396         [M, N, 1]
397     ]).T # Transpuesta para facilitar operaciones matriciales

```

```

1398 # Transformar las esquinas de la imagen
1399 transformed_corners = H @ corners
1400 transformed_corners /= transformed_corners[2, :] # Normalizar
1401
1402 # Calcular los límites de la nueva imagen
1403 x_min, y_min = np.min(transformed_corners[:2, :], axis=1)
1404 x_max, y_max = np.max(transformed_corners[:2, :], axis=1)
1405
1406 # Calcular el nuevo tamaño de la imagen transformada
1407 Np = int(np.ceil(y_max - y_min + 1))
1408 Mp = int(np.ceil(x_max - x_min + 1))
1409
1410 # Crear una imagen vacía para almacenar la imagen transformada
1411 imagen_transformada = np.zeros((Np, Mp, L), dtype=np.uint8)
1412
1413 # Calcular la matriz de transformación inversa
1414 H_inv = np.linalg.inv(H)
1415
1416 # Recorrer la imagen transformada pixel por pixel
1417 for yp in range(Np):
1418     for xp in range(Mp):
1419         # Coordenadas ajustadas (en coordenadas homogéneas)
1420         p_p = np.array([xp + x_min - 1, yp + y_min - 1, 1])
1421
1422         # Aplicar la transformación inversa
1423         p = H_inv @ p_p
1424         x_o = int(round(p[0] / p[2]))
1425         y_o = int(round(p[1] / p[2]))
1426
1427         # Verificar si las coordenadas están dentro de los límites de la imagen original
1428         if 1 <= x_o <= M and 1 <= y_o <= N:
1429             for c in range(L): # Recorrer cada canal
1430                 imagen_transformada[yp, xp, c] = I[y_o - 1, x_o - 1, c] # Ajustar a 0-indexed
1431
1432 return imagen_transformada

```


Bibliografía

- [1] Saeed Anwar, Chongyi Li, and Fatih Porikli. Image colorization: A survey and dataset. *arXiv preprint arXiv:2008.10774*, 2020.
- [2] Jaakko Astola and Pauli Kuosmanen. *Nonlinear Image Processing*. Academic Press, 1990.
- [3] Ronald N. Bracewell. The fourier transform and its applications. *IEEE Spectrum*, 2(12):63–70, 1965.
- [4] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of graphics tools*, 12(2):13–21, 2007.
- [5] Wilhelm Burger and Mark J. Burge. *Principles of Digital Image Processing: Core Algorithms*. Springer, 2009.
- [6] Wilhelm Burger and Mark J. Burge. *Principles of Digital Image Processing: Core Algorithms*. Undergraduate Topics in Computer Science. Springer, London, 2009.
- [7] Martin Čadík. Perceptual evaluation of color-to-grayscale image conversions. *Computer Graphics Forum*, 27(7):1745–1754, 2008.
- [8] Doru Coltuc, Patrice Bolon, and Jean-Marc Chassery. Exact histogram specification. *IEEE Transactions on Image Processing*, 15(5):1143–1152, 2006.
- [9] Commission Internationale de l’Éclairage (CIE). *Commission Internationale de l’Éclairage Proceedings, 1931*. Cambridge University Press, 1932. Definición del observador estándar CIE 1931.
- [10] Commission Internationale de l’Éclairage (CIE). *CIE 015:2004 Colorimetry, 3rd Edition*. CIE Central Bureau, Vienna, 2004.
- [11] Edward R. Dougherty. *An Introduction to Morphological Image Processing*. SPIE Optical Engineering Press, 1992.
- [12] Arrow Electronics. What is a bayer filter? bayer color filter array explained. *Arrow.com*, 2017.
- [13] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson, 4th edition, 2018.
- [14] Todd Gustavson. *Camera: A History of Photography from Daguerreotype to Digital*. Sterling Signature, 2009. Creado en colaboración con la George Eastman House.
- [15] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, 2nd edition, 2003.
- [16] Licheng He, Yuyan Chao, Kenichi Suzuki, and Kesheng Wu. Fast connected-component labeling. *Pattern Recognition*, 42(9):1977–1987, 2009.

- [17] Alkin Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.
- [18] Quang Huynh-Thu and Mohammed Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics Letters*, 44(13):800–801, 2008.
- [19] JAI Inc. Improve color accuracy & more with a 3-ccd digital camera, 2016. Imagen comparativa entre sistemas 3CCD y 1CCD publicada en el blog de 1stVision.
- [20] Gerald H. Jacobs. Evolution of colour vision in mammals. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1531):2957–2967, 2009.
- [21] Bernd Jähne. *Digital Image Processing: Concepts, Algorithms, and Scientific Applications*. Springer, Berlin, Heidelberg, 6th edition, 2005.
- [22] Philippe Kahn. Birth of the camera phone, 1997. Consultado el 4 de abril de 2025.
- [23] Jagat Narain Kapur, Prasanna K Sahoo, and Andrew KC Wong. A new method for gray-level picture thresholding using the entropy of the histogram. *Computer vision, graphics, and image processing*, 29(3):273–285, 1985.
- [24] David King. *The Commissar Vanishes: The Falsification of Photographs and Art in Stalins Russia*. Metropolitan Books, 1997. Documenta y analiza casos reales de manipulación fotográfica durante el régimen de Stalin.
- [25] Russell A. Kirsch and Larry G. Roberts. Experiments in processing pictorial information with a digital computer. *Proceedings of the Eastern Joint Computer Conference*, pages 221–229, 1957.
- [26] Shaohui Liu. Two decades of colorization and decolorization for images and videos. *arXiv preprint arXiv:2204.13322*, 2022.
- [27] Zhipeng Lu, Xiaojie Wang, Yuan Yuan, Qiang Yang, and Nenghai Yu. Color-to-grayscale: Does the method matter in image recognition? *Pattern Recognition Letters*, 31(12):1665–1670, 2010.
- [28] Dan Margulis. *Professional Photoshop: The Classic Guide to Color Correction*. Peachpit Press, 5th edition, 2006.
- [29] Elaine N. Marieb and Katja Hoehn. *Anatomía y fisiología humanas*. Pearson Educación, 10^a edición, 2018.
- [30] James Clerk Maxwell. Xviii. experiments on colour, as perceived by the eye, with remarks on colour-blindness. *Transactions of the Royal Society of Edinburgh*, 21:275–298, 1857.
- [31] James Clerk Maxwell. The first colour photographic image. Royal Institution demonstration, 17 May 1861, 1861. Clerk Maxwell Foundation https://www.clerkmaxwellfoundation.org/html/first_colour_photographic_image.html.
- [32] John Miano. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. Addison-Wesley Professional, 1999.
- [33] Soma Mukherjee and Debasis Das. Histogram equalization techniques for contrast enhancement in medical images: A review. *Procedia Computer Science*, 167:26–36, 2019.
- [34] NASA. Apollo 11 photograph - buzz aldrin, 1969. Consultado el 4 de abril de 2025.
- [35] NASA. Pillars of creation, 1995. Consultado el 4 de abril de 2025.
- [36] Beaumont Newhall. *The History of Photography: From 1839 to the Present*. The Museum of Modern Art, 1982.

- [37] W Niblack. An introduction to digital image processing, 1986.
- [38] Joseph Nicéphore Niépce. View from the window at le gras, 1826. Consultado el 4 de abril de 2025.
- [39] Nobel Prize Organization. The dual nature of light as reflected in the nobel archives, 1998.
- [40] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.
- [41] Tekla S. Perry. The first digital camera was the size of a toaster. *IEEE Spectrum*, 2017.
- [42] Judith M. S. Prewitt. Object enhancement and extraction. In Bernard S. Lipkin and Azriel Rosenfeld, editors, *Picture Processing and Psychopictorics*, pages 75–149. Academic Press, 1970.
- [43] T. Pun. Entropic thresholding: a new approach. *Computer Graphics and Image Processing*, 16(3):210–239, 1981.
- [44] TW Ridler and S Calvard. Picture thresholding using an iterative selection method. *IEEE Trans. Syst. Man Cybern.*, 8(8):630–632, 1978.
- [45] Wilhelm Conrad Röntgen. On a new kind of rays. *Proceedings of the Würzburg Physical-Medical Society*, 1895. Consultado el 4 de abril de 2025.
- [46] Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM (JACM)*, 13(4):471–494, 1966.
- [47] Roy Rosenzweig. Can history be open source? wikipedia and the future of the past. *The Journal of American History*, 93(1):117–146, 2006. Incluye una discusión sobre la manipulación histórica de imágenes.
- [48] Jaakko Sauvola and Matti Pietikäinen. Adaptive document image binarization. *Pattern recognition*, 33(2):225–236, 2000.
- [49] NASA Science. Visible light, 2023.
- [50] Gaurav Sharma, editor. *Digital Color Imaging Handbook*. CRC Press, Boca Raton, FL, 2003.
- [51] Sam Shere. Explosion of the hindenburg, lakehurst, nj, 1937. Consultado el 4 de abril de 2025.
- [52] Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. Presented at Stanford Artificial Intelligence Project (SAIL), 1968. Unpublished internal document.
- [53] Pierre Soille. *Morphological Image Analysis: Principles and Applications*. Springer, 2nd edition, 2003.
- [54] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, London, 2010.
- [55] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846. IEEE, 1998.
- [56] William Ardila Urueña, Jimmy Alexander Cortés, Jairo Alberto Mendoza Vargas, and Jimmy Alexander Osorio. Técnicas alternativas para la conversión de imágenes a color a escala de grises en el tratamiento digital de imágenes. *Scientia et Technica*, 1(47):207–212, 2011.
- [57] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.
- [58] Günther Wyszecki and Walter S. Stiles. Color science: Concepts and methods, quantitative data and formulae.

Wiley Series in Pure and Applied Optics, 1982.