UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

Inteligencia Artificial

Ing. Juancarlos Santana Huamán
jsantana@ucss.edu.pe

Filtros en Procesamiento de Imágenes

UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

- Los filtros son operaciones que modifican los píxeles de una imagen basándose en los valores de sus vecinos. Se utilizan para realzar características, reducir ruido o extraer información.
- **Tipos principales:**
  - Filtros de suavizado: Reducen ruido y detalles finos
    - Media: Promedia los valores de píxeles vecinos
    - Gaussiano: Da más peso a los píxeles centrales (distribución normal)
  - Filtros de realce: Resaltan características específicas
    - Laplaciano: Detecta áreas de cambio rápido (bordes)
    - Prewitt/Sobel: Detectan bordes en direcciones específicas

Kernel/Matriz de convolución: Pequeña matriz que define cómo se combinan los píxeles vecinos.

Filtros de Suavizado (Paso Bajo)

- Filtro de Media:
  - Kernel: $K = 1/(m \times n) \times$ [matriz de unos]
  - Efecto: Promedia los valores de los píxeles vecinos
  - Reduce ruido, pero causa desenfoque
- Filtro Gaussiano:
  - Kernel basado en distribución normal 2D:

    $$G(x,y) = (1/(2\pi\sigma^2)) \times \exp(-(x^2+y^2)/(2\sigma^2))$$
  - Parámetro σ controla el grado de suavizado
  - Preserva mejor los bordes que el filtro de media

Filtros de Realce (Paso Alto)

- Filtro Laplaciano:
  - Aproxima la segunda derivada: $\nabla^2 I = \partial^2 I/\partial x^2 + \partial^2 I/\partial y^2$
  - Kernel típico:

    [0 -1 0]

    [-1 4 -1]

    [0 -1 0]
  - Resalta regiones de cambio rápido de intensidad
- Filtros de Gradiente (Sobel, Prewitt):
  - Aproximan las primeras derivadas en direcciones x e y
  - Sobel:

    Gx = [-1 0 1; -2 0 2; -1 0 1]

    Gy = [-1 -2 -1; 0 0 0; 1 2 1]
  - Magnitud del gradiente: $|\nabla I| = \sqrt{(Gx^2 + Gy^2)}$
  - Dirección del gradiente: $\theta = \text{atan2}(Gy, Gx)$

Detección de Bordes

- Identificar puntos donde la intensidad de la imagen cambia abruptamente, indicando transiciones entre objetos o regiones.
- Métodos principales:
  - Operadores de gradiente: Calculan la derivada de la imagen
    - Sobel: Aproximación discreta de la derivada en direcciones x e y
    - Prewitt: Similar a Sobel pero con diferentes pesos
  - Canny Edge Detector: Algoritmo de múltiples etapas:
    - Reducción de ruido con filtro Gaussiano
    - Cálculo del gradiente de intensidad
    - Supresión de no-máximos (afinar bordes)
    - Umbralización con histéresis (bordes fuertes y débiles)

Operadores de Gradiente

- Fundamento Matemático
  - Los operadores de gradiente calculan la derivada direccional de la función de intensidad de la imagen I(x,y). El gradiente vectorial se define como:

    $\nabla I = [\partial I/\partial x, \partial I/\partial y]^T$
  - La magnitud del gradiente indica la tasa de cambio de intensidad:

    $|\nabla I| = \sqrt{[(\partial I/\partial x)^2 + (\partial I/\partial y)^2]}$
  - La dirección del gradiente indica la orientación del cambio máximo:

    $\theta = \text{atan2}(\partial I/\partial y, \partial I/\partial x)$

Operador de Sobel

- Kernels de Sobel
  - El operador de Sobel utiliza dos kernels ortogonales 3×3:
  - Dirección x (vertical edges):

    $$Sx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

  - Dirección y (horizontal edges):

    $$Sy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Implementación Matemática

  $$Gx = I * Sx = \sum\sum I(x+i, y+j) \times Sx(i,j)$$

  $$Gy = I * Sy = \sum\sum I(x+i, y+j) \times Sy(i,j)$$

- Magnitud:

  $$M(x,y) = \sqrt{(Gx^2 + Gy^2)}$$

- Dirección:

  $$\theta(x,y) = \text{atan2}(Gy, Gx)$$

- Propiedades de Sobel
  - Los pesos [1, 2, 1] suavizan en la dirección perpendicular a la derivada
  - Mayor sensibilidad a bordes diagonales que operadores simples
  - Implementación computacionalmente eficiente

Operador de Prewitt

- Kernels de Prewitt
  - Dirección x:

    Px = [-1 0 1]

    [-1 0 1]

    [-1 0 1]
  - Dirección y:

    Py = [-1 -1 -1]

    [0 0 0]

    [1 1 1]
- Comparación Sobel vs Prewitt
  - Sobel: Da más peso a los píxeles centrales (menos sensible al ruido)
  - Prewitt: Todos los píxeles tienen igual peso (más sensible al ruido)
  - Sobel generalmente produce bordes más suaves
  - Prewitt es más rápido computacionalmente

Detector de Bordes de Canny

- John Canny (1986) definió tres criterios para un detector de bordes óptimo:
  - Buena detección: Mínima probabilidad de falsos positivos y falsos negativos
  - Buena localización: Los bordes detectados deben estar lo más cerca posible de los bordes reales
  - Respuesta única: Solo una respuesta por borde real
- Etapa 1: Reducción de Ruido con Filtro Gaussiano
  - Filtro Gaussiano 2D:

    $G(x,y) = (1/(2\pi\sigma^2)) \times \exp(-(x^2 + y^2)/(2\sigma^2))$
  - Aplicación:

    $I\_suavizada(x,y) = \sum\sum I(u,v) \times G(x-u, y-v)$
  - Propiedades:
    - σ controla el grado de suavizado (compromiso entre supresión de ruido y preservación de bordes)
    - Mayor $\sigma \rightarrow$ mayor suavizado pero bordes más difusos

Detector de Bordes de Canny

- Etapa 2: Cálculo del Gradiente de Intensidad
  - Se aplican operadores de gradiente optimizados:

    $G_x = I\_suavizada * D_x$

    $G_y = I\_suavizada * D_y$
  - Donde Dx y Dy son derivadas del Gaussiano:

    $D_x = \partial G/\partial x$, $D_y = \partial G/\partial y$
  - Magnitud y dirección:

    $M(x,y) = \sqrt{(G_x^2 + G_y^2)}$

    $\theta(x,y) = \text{atan2}(G_y, G_x)$ (cuantizada a 0°, 45°, 90°, 135°)

Detector de Bordes de Canny

- Etapa 3: Supresión de No-Máximos
  - Afinar los bordes eliminando píxeles que no son máximos locales en la dirección del gradiente.
  - Algoritmo
    - Para cada píxel (x,y):
      - Redondear θ(x,y) a la dirección más cercana (0°, 45°, 90°, 135°)
      - Comparar M(x,y) con sus dos vecinos en la dirección de θ
      - Si M(x,y) no es el máximo entre sus vecinos, suprimirlo (M(x,y) = 0)
  - Ejemplo para θ = 0°:

    Comparar con (x,y-1) y (x,y+1)
  - Ejemplo para θ = 45°:

    Comparar con (x-1,y+1) y (x+1,y-1)
- Etapa 4: Umbralización con Histéresis
  - Problema de Umbral Simple
  - Un único umbral produce:
    - Umbral alto: bordes fragmentados
    - Umbral bajo: bordes falsos por ruido
  - Solución: Doble Umbral
    - Umbral alto (T_high): Solo bordes fuertes
    - Umbral bajo (T_low): Bordes débiles potenciales

Detector de Bordes de Canny

- Algoritmo de Histéresis
  - Clasificación inicial:

    $M(x,y) \geq T_{high}$ → Borde fuerte (siempre se conserva)

    $T_{low} \leq M(x,y) < T_{high}$ → Borde débil (se evalúa)

    $M(x,y) < T_{low}$ → Se descarta
  - Conectividad:
    - Para cada borde débil, verificar si está conectado a algún borde fuerte
    - Si está conectado → Conservar como borde
    - Si no está conectado → Descartar
- Implementación Práctica
  - Usualmente: $T_{high} = 2 \times T_{low}$
  - Valores típicos: $T_{low} = 0.05 \times \max(M)$, $T_{high} = 0.15 \times \max(M)$
  - La conectividad se verifica mediante vecindarios 8-conectados

Detector de Bordes de Canny

- Ventajas del Detector de Canny
  - Baja tasa de error: Minimiza falsos positivos y negativos
  - Buena localización: Los bordes están bien posicionados
  - Respuesta única: Un solo píxel de ancho por borde
  - Robustez al ruido: Gracias al suavizado Gaussianoy la histéresis
- Consideraciones de Implementación
  - Elección de parámetros:
    - σ: Controla el suavizado (típico: 1.0-2.0)
    - T_low, T_high: Ajustan la sensibilidad
    - Tamaño del kernel Gaussian: Generalmente 3×3 o 5×5
  - Complejidad computacional:
    - O(n) para convolución Gaussian
    - O(n) para cálculo de gradiente
    - O(n) para supresión de no-máximos
    - O(n) para umbralización con histéresis
- Este enfoque de múltiples etapas hace del detector de Canny uno de los métodos más efectivos y ampliamente utilizados en la detección de bordes en visión por computadora.

Detección de Esquinas

- Identificar puntos donde la intensidad cambia significativamente en múltiples direcciones.
- Algoritmos principales:
  - Harris Corner Detection:
    - Calcula la matriz de autocorrelación
    - Mide el cambio en todas las direcciones
    - Usa función de respuesta para identificar esquinas
  - Shi-Tomasi Corner Detection: Variante de Harris que usa el mínimo valor propio

Detección de Esquinas

- Punto donde la intensidad cambia significativamente en múltiples direcciones. A diferencia de los bordes (cambios predominantemente unidireccionales), las esquinas tienen variaciones bidireccionales.
- Algoritmo de Harris Corner Detection
  - Matriz de Autocorrelación
    - Para una ventana W y un desplazamiento (u,v):

      $E(u,v) = \sum\sum [I(x+u,y+v) - I(x,y)]^2 \approx \sum\sum [u\ v]\ M\ [u;\ v]$
    - Donde M es la matriz de segundo momento:

      $M = \sum\sum [Ix^2\ IxIy;\ IxIy\ Iy^2]$
- Función de Respuesta de Esquina

  $R = \det(M) - k \times \text{trace}(M)^2$
  - Donde:
    - $\det(M) = \lambda_1\lambda_2$ (producto de valores propios)
    - $\text{trace}(M) = \lambda_1 + \lambda_2$ (suma de valores propios)
    - k: constante empírica (~0.04-0.06)
- Interpretación de R:
  - $R > 0$: Esquina (ambos λ grandes y similares)
  - $R \approx 0$: Región plana (ambos λ pequeños)
  - $R < 0$: Borde (un λ grande, otro pequeño)

Detección de Esquinas

- Algoritmo de Shi-Tomasi
  - Variante que usa:

    $R = \min(\lambda_1, \lambda_2)$
  - Esquina cuando $\min(\lambda_1, \lambda_2) >$ umbral. Generalmente más robusto que Harris.

Detección de Líneas y Curvas

- Identificar formas geométricas específicas en imágenes.
- Métodos principales:
  - Transformada de Hough:
    - Para líneas: Representa líneas en espacio de parámetros (ρ, θ)
    - Para círculos: Representa círculos en espacio (x, y, radio)
    - Generalizada: Para otras formas geométricas
  - Detección basada en contornos: Usa información de bordes para encontrar formas

Detección de Líneas y Curvas

Transformada de Hough. Transforma puntos del espacio de la imagen al espacio de parámetros de la forma geométrica buscada.

- Para Líneas Rectas
  - Representación normal: $\rho = x \cdot \cos\theta + y \cdot \sin\theta$
  - Donde:
    - ρ: distancia desde el origen a la línea
    - θ: ángulo del vector normal
  - Algoritmo:
    - Discretizar el espacio (ρ, θ) en celdas (acumulador)
    - Para cada punto de borde (x,y) y cada θ, calcular ρ
    - Incrementar el acumulador en (ρ, θ)
    - Buscar máximos en el acumulador

Detección de Líneas y Curvas

- Para Círculos
  - Ecuación: $(x-a)^2 + (y-b)^2 = r^2$
  - Espacio de parámetros tridimensional: (a, b, r)
  - Computacionalmente costoso para múltiples radios
- Transformada de Hough Generalizada
  - Para formas arbitrarias descritas por:
    - $F(x, y, p_1, p_2, ..., p_n) = 0$
  - Donde $p_1...p_n$ son los parámetros de la forma.

Detección Basada en Contornos

- Algoritmo de Marching Squares
  - Versión 2D de Marching Cubes
  - Divide la imagen en cuadrados 2×2
  - Para cada cuadrado, determina la configuración de píxeles sobre/ bajo umbral
  - Conecta puntos para formar contornos
- Aproximación de Polígonos
  - Algoritmo de Ramer-Douglas-Peucker
  - Simplifica curvas complejas mediante aproximación poligonal
  - Parámetro ε controla la precisión de la aproximación

Aplicación con Python

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import feature, transform
from scipy import ndimage

# Configurar matplotlib
plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams['image.cmap'] = 'gray'
```

UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

```python
def ejercicio_filtros():
    # Crear imagen de ejemplo con ruido
    imagen = np.zeros((100, 100))
    imagen[20:80, 20:80] = 1  # Cuadrado blanco
    # Añadir ruido
    ruido = np.random.normal(0, 0.1, imagen.shape)
    imagen_ruidosa = np.clip(imagen + ruido, 0, 1)
    # Aplicar diferentes filtros
    filtro_media = cv2.blur(imagen_ruidosa, (5, 5))
    filtro_gaussiano = cv2.GaussianBlur(imagen_ruidosa, (5, 5), 0)
    filtro_mediana = cv2.medianBlur((imagen_ruidosa * 255).astype(np.uint8), 5)
    # Mostrar resultados
    fig, axes = plt.subplots(2, 2)
    axes[0,0].imshow(imagen_ruidosa)
    axes[0,0].set_title('Imagen con ruido')
    axes[0,1].imshow(filtro_media)
    axes[0,1].set_title('Filtro de Media')
    axes[1,0].imshow(filtro_gaussiano)
    axes[1,0].set_title('Filtro Gaussiano')
    axes[1,1].imshow(filtro_mediana)
    axes[1,1].set_title('Filtro de Mediana')
    plt.tight_layout()
    plt.show()
ejercicio_filtros()
```

```python
def ejercicio_deteccion_bordes():
    # Crear imagen con diferentes formas
    imagen = np.zeros((200, 200))
    cv2.rectangle(imagen, (30, 30), (80, 80), 1, -1)
    cv2.circle(imagen, (150, 150), 30, 1, -1)
    cv2.line(imagen, (100, 30), (180, 180), 1, 3)
    # Añadir ruido para hacerlo más realista
    ruido = np.random.normal(0, 0.05, imagen.shape)
    imagen = np.clip(imagen + ruido, 0, 1)
    # Detectar bordes con diferentes métodos
    bordes_sobelx = cv2.Sobel(imagen, cv2.CV_64F, 1, 0, ksize=3)
    bordes_sobely = cv2.Sobel(imagen, cv2.CV_64F, 0, 1, ksize=3)
    bordes_sobel = np.sqrt(bordes_sobelx**2 + bordes_sobely**2)
    bordes_laplace = cv2.Laplacian(imagen, cv2.CV_64F)
    bordes_canny = feature.canny(imagen, sigma=1.0)
    # Mostrar resultados
    fig, axes = plt.subplots(2, 3)
    axes[0,0].imshow(imagen)
    axes[0,0].set_title('Imagen Original')
    axes[0,1].imshow(bordes_sobelx)
    axes[0,1].set_title('Sobel X')
    axes[0,2].imshow(bordes_sobely)
    axes[0,2].set_title('Sobel Y')
    axes[1,0].imshow(bordes_sobel)
    axes[1,0].set_title('Sobel Combinado')
    axes[1,1].imshow(bordes_laplace)
    axes[1,1].set_title('Laplaciano')
    axes[1,2].imshow(bordes_canny)
    axes[1,2].set_title('Canny')
    plt.tight_layout()
    plt.show()
ejercicio_deteccion_bordes()
```

UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

```python
def ejercicio_deteccion_esquinas():
    # Crear imagen con esquinas
    imagen = np.zeros((100, 100))

    # Crear patrones que generen esquinas
    imagen[20:40, 20:80] = 1  # Rectángulo horizontal
    imagen[40:80, 60:80] = 1  # Rectángulo vertical

    # Detectar esquinas con Harris
    imagen_float = np.float32(imagen)
    esquinas_harris = cv2.cornerHarris(imagen_float, 2, 3, 0.04)

    # Umbralizar para obtener puntos de esquina
    esquinas_harris = cv2.dilate(esquinas_harris, None)
    umbral = 0.01 * esquinas_harris.max()
    imagen_esquinas = imagen.copy()
    imagen_esquinas[esquinas_harris > umbral] = 0.5  # Marcar esquinas

    # Detectar esquinas con Shi-Tomasi
    esquinas = cv2.goodFeaturesToTrack(imagen_float, 25, 0.01, 10)
    esquinas = np.int0(esquinas)

    imagen_shi_tomasi = imagen.copy()
    for i in esquinas:
        x, y = i.ravel()
        cv2.circle(imagen_shi_tomasi, (x, y), 3, 0.5, -1)

    # Mostrar resultados
    fig, axes = plt.subplots(1, 3)
    axes[0].imshow(imagen)
    axes[0].set_title('Imagen Original')

    axes[1].imshow(imagen_esquinas)
    axes[1].set_title('Harris Corner Detection')

    axes[2].imshow(imagen_shi_tomasi)
    axes[2].set_title('Shi-Tomasi Corner Detection')

    plt.tight_layout()
    plt.show()

ejercicio_deteccion_esquinas()
```

```python
def ejercicio_deteccion_lineas():
    # Crear imagen con líneas
    imagen = np.zeros((200, 200))

    # Dibujar líneas en diferentes ángulos
    cv2.line(imagen, (20, 20), (180, 20), 1, 2)  # Horizontal
    cv2.line(imagen, (20, 50), (180, 100), 1, 2)  # Diagonal
    cv2.line(imagen, (20, 150), (20, 50), 1, 2)   # Vertical

    # Añadir ruido
    ruido = np.random.normal(0, 0.03, imagen.shape)
    imagen = np.clip(imagen + ruido, 0, 1)

    # Detectar bordes con Canny
    bordes = feature.canny(imagen, sigma=1.0)

    # Aplicar Transformada de Hough para líneas
    h, theta, d = transform.hough_line(bordes)
    lineas = transform.hough_line_peaks(h, theta, d, threshold=0.7 * np.max(h))
```

```
    # Dibujar líneas detectadas
    imagen_resultado = np.copy(imagen)

    for _, angulo, dist in zip(*lineas):
        y0 = (dist - 0 * np.cos(angulo)) / np.sin(angulo)
        y1 = (dist - imagen.shape[1] * np.cos(angulo)) / np.sin(angulo)
        cv2.line(imagen_resultado, (0, int(y0)), (imagen.shape[1], int(y1)), 0.7, 2)

    # Mostrar resultados
    fig, axes = plt.subplots(1, 3)
    axes[0].imshow(imagen)
    axes[0].set_title('Imagen Original')

    axes[1].imshow(bordes)
    axes[1].set_title('Bordes Detectados')

    axes[2].imshow(imagen_resultado)
    axes[2].set_title('Líneas Detectadas')

    plt.tight_layout()
    plt.show()

ejercicio_deteccion_lineas()
```

Ejercicio Completo

```python
def ejercicio_completo():
    # Cargar imagen real o crear una más compleja
    imagen = np.zeros((300, 300))

    # Crear múltiples formas
    cv2.rectangle(imagen, (50, 50), (100, 100), 1, -1)  # Cuadrado
    cv2.circle(imagen, (200, 100), 40, 1, -1)           # Círculo
    cv2.line(imagen, (150, 200), (250, 250), 1, 3)      # Línea

    # Añadir ruido
    ruido = np.random.normal(0, 0.05, imagen.shape)
    imagen = np.clip(imagen + ruido, 0, 1)

    # Pipeline completo de procesamiento
    # 1. Filtrado para reducir ruido
    imagen_filtrada = cv2.GaussianBlur(imagen, (5, 5), 0)
```

```python
# 2. Detección de bordes
bordes = feature.canny(imagen_filtrada, sigma=1.5)

# 3. Detección de esquinas
imagen_float = np.float32(imagen_filtrada)
esquinas_harris = cv2.cornerHarris(imagen_float, 2, 3, 0.04)
esquinas_harris = cv2.dilate(esquinas_harris, None)
umbral = 0.01 * esquinas_harris.max()

# 4. Detección de líneas
h, theta, d = transform.hough_line(bordes)
lineas = transform.hough_line_peaks(h, theta, d, threshold=0.6 * np.max(h))

# Crear imagen de resultados
resultado = np.zeros_like(imagen)
resultado[bordes] = 0.3  # Bordes en gris claro

# Marcar esquinas
resultado[esquinas_harris > umbral] = 0.6  # Esquinas en gris medio
```

UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

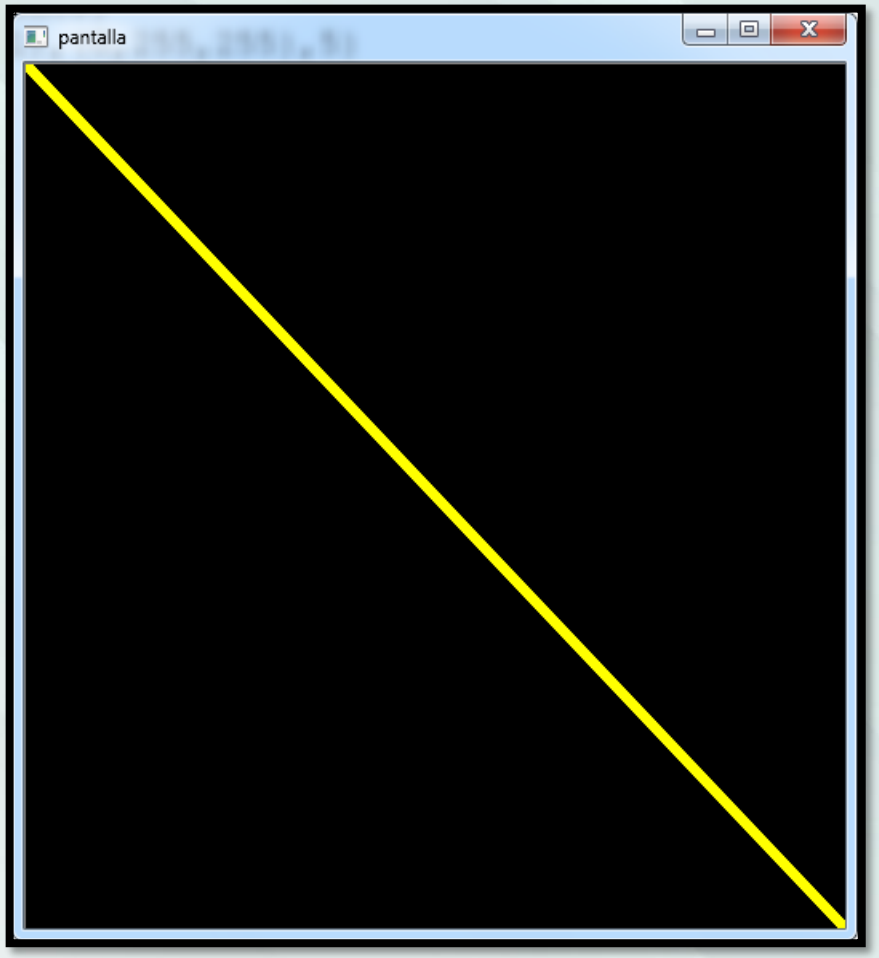
```python
# Dibujar líneas detectadas
for _, angulo, dist in zip(*lineas):
    y0 = (dist - 0 * np.cos(angulo)) / np.sin(angulo)
    y1 = (dist - imagen.shape[1] * np.cos(angulo)) / np.sin(angulo)
    # Convertir a coordenadas enteras para dibujar
    x0, y0 = 0, int(y0)
    x1, y1 = imagen.shape[1], int(y1)
    cv2.line(resultado, (x0, y0), (x1, y1), 0.9, 2)  # Líneas en blanco

# Mostrar pipeline completo
fig, axes = plt.subplots(2, 3)
axes[0,0].imshow(imagen)
axes[0,0].set_title('Imagen Original')

axes[0,1].imshow(imagen_filtrada)
axes[0,1].set_title('Imagen Filtrada')

axes[0,2].imshow(bordes)
axes[0,2].set_title('Bordes Detectados')

axes[1,0].imshow(esquinas_harris > umbral)
axes[1,0].set_title('Esquinas Detectadas')
```

```
    axes[1,1].imshow(h, extent=[np.rad2deg(theta[-1]), np.rad2deg(theta[0]),
                                d[-1], d[0]], aspect=1/1.5)
    axes[1,1].set_title('Transformada de Hough')
    axes[1,1].set_xlabel('Ángulo (grados)')
    axes[1,1].set_ylabel('Distancia (píxeles)')

    axes[1,2].imshow(resultado)
    axes[1,2].set_title('Resultado Final')

    plt.tight_layout()
    plt.show()

ejercicio_completo()
```

Dibujar una línea:

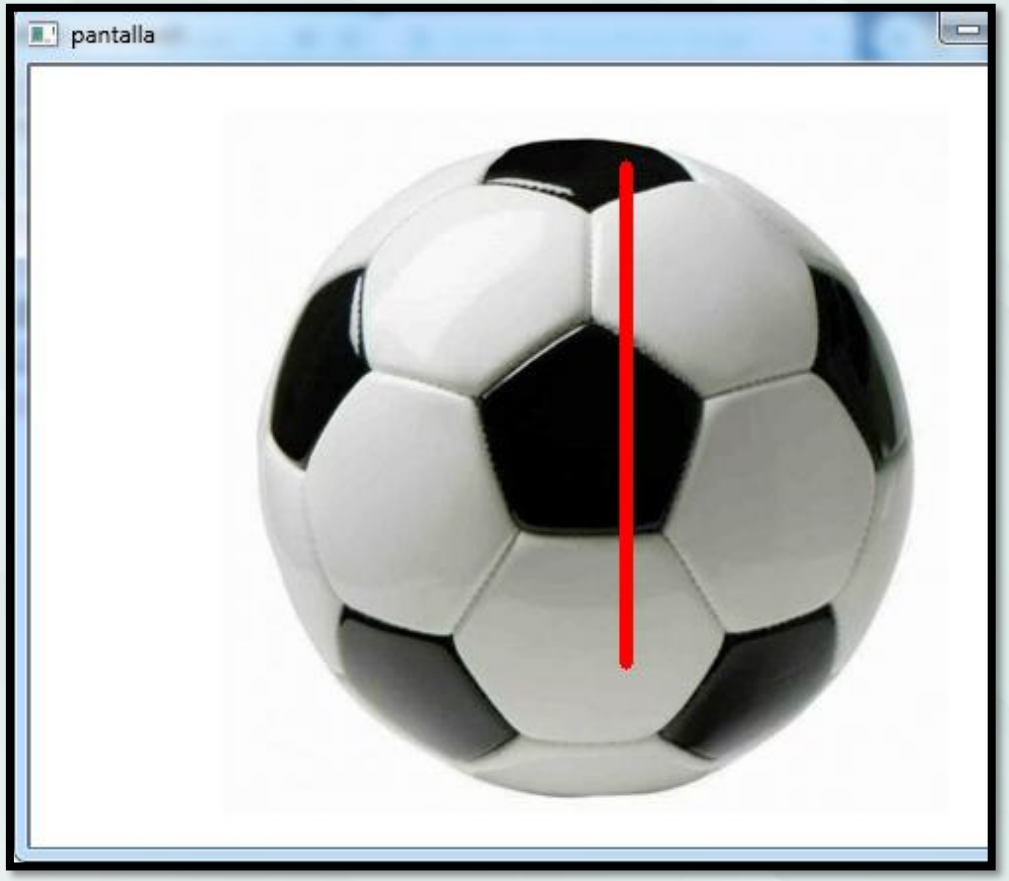
Para dibujar una línea, debe pasar las coordenadas iniciales y finales de la línea. Crearemos una imagen en negro y dibujaremos una línea azul en ella desde las esquinas superior izquierda a inferior derecha.

```
import numpy as np
import cv2

img = np.zeros((512,512,3), np.uint8)
img = cv2.line(img,(0,0),(511,511),(0,255,255),5)
cv2.imshow('pantalla',img)
```



Dibujar una línea en una imagen:

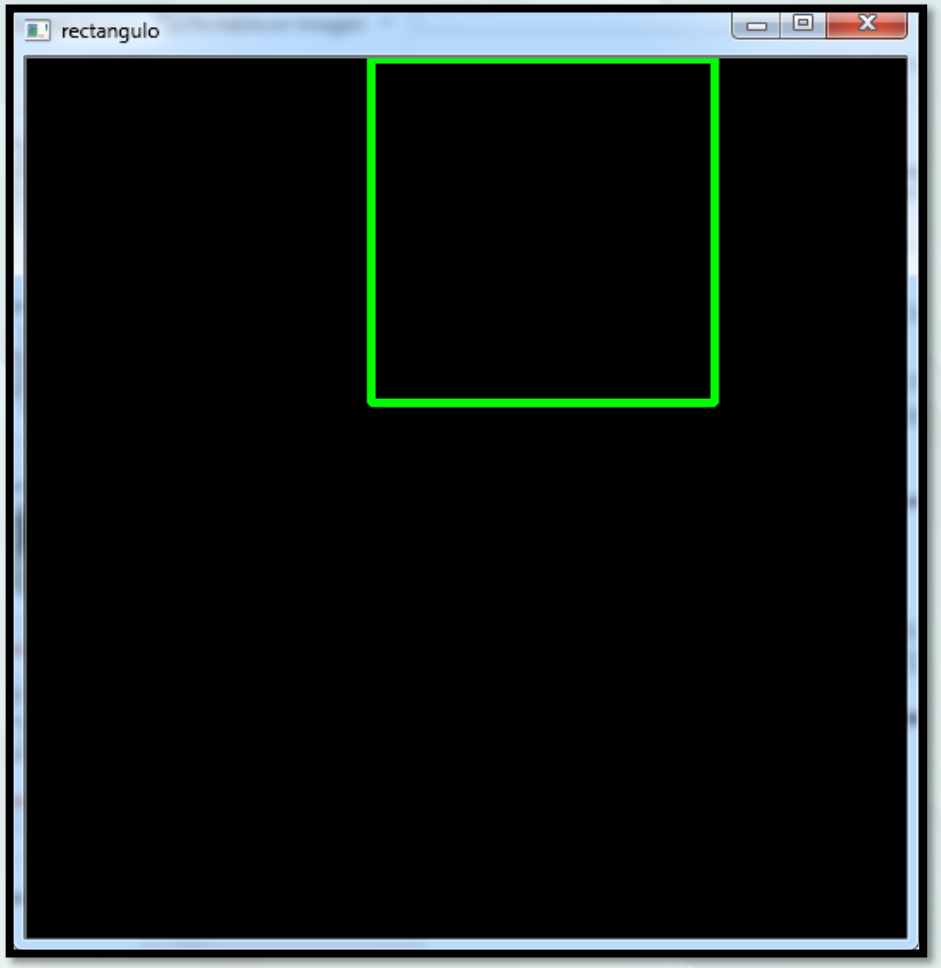
```
import numpy as np
import cv2

img = cv2.imread('pelota.jpg')
img = cv2.line(img,(300,50),(300,300),(0,0,255),5)
cv2.imshow('pantalla',img)
```



Dibujar un Rectángulo:

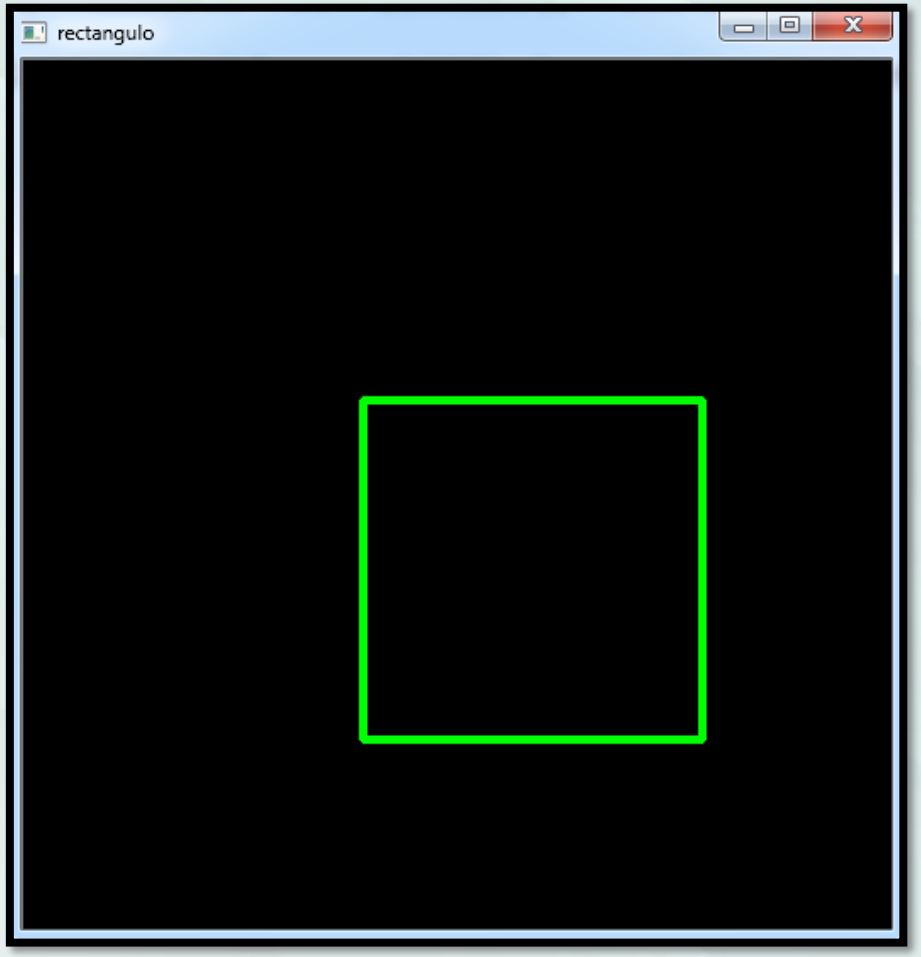
Para dibujar un rectángulo, necesita la esquina superior izquierda y la esquina inferior derecha del rectángulo. Esta vez dibujaremos un rectángulo verde en la esquina superior derecha de la imagen.

```
import numpy as np
import cv2

img = np.zeros((512,512,3), np.uint8)
img = cv2.rectangle(img,(200,0),(400,200),(0,255,0),3)
cv2.imshow('rectangulo',img)
```



Dibujar un Rectángulo:

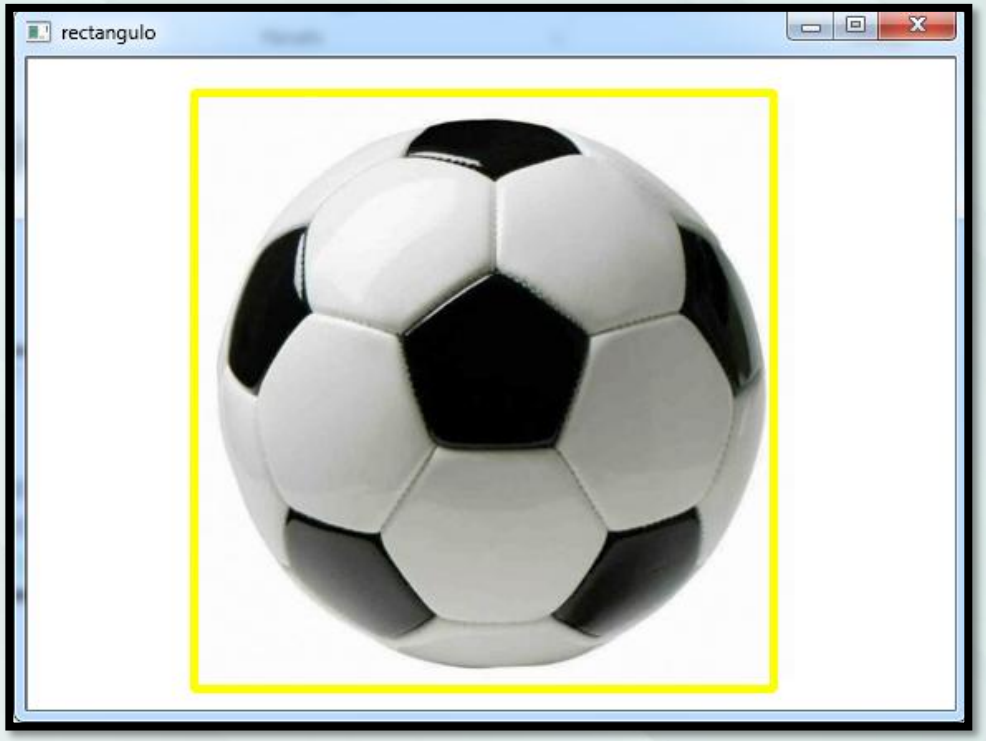
```
import numpy as np
import cv2

img = np.zeros((512,512,3), np.uint8)
img = cv2.rectangle(img,(200,200),(400,400),(0,255,0),3)
cv2.imshow('rectangulo',img)
```



UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

Dibujar un Rectángulo en una Figura:

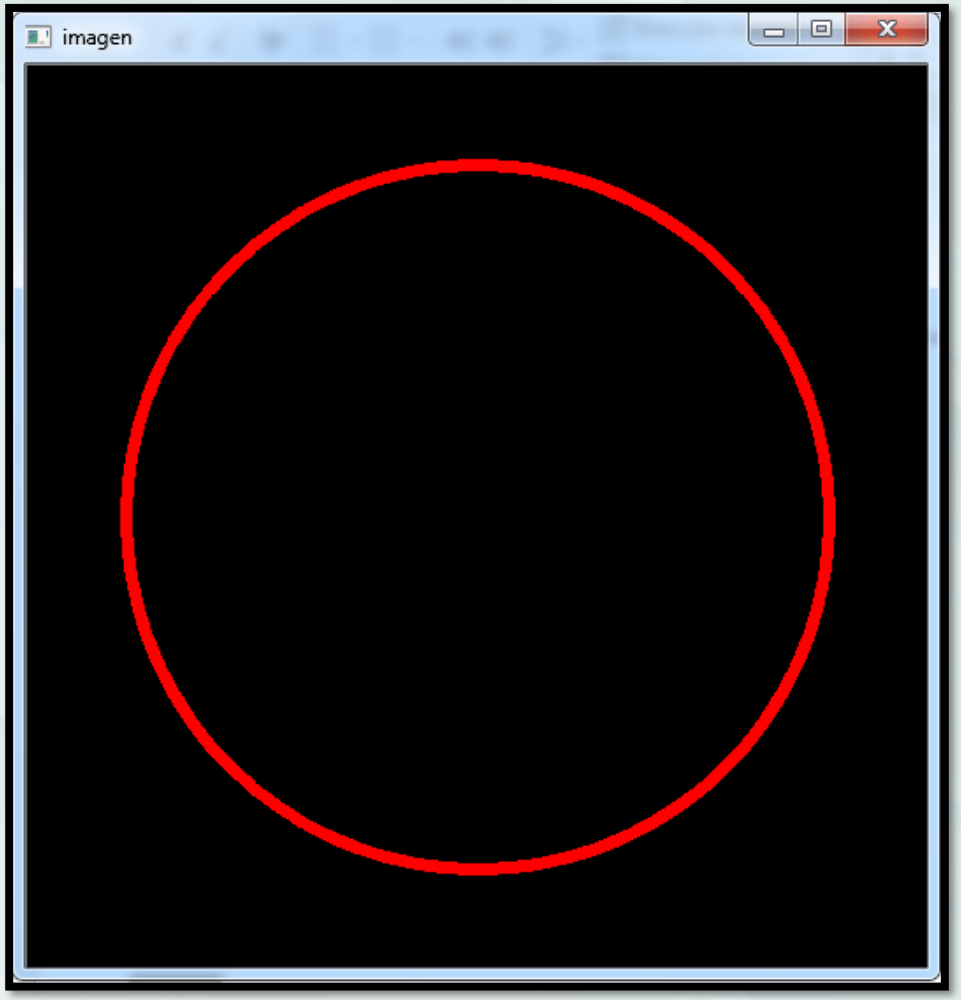
```
import numpy as np
import cv2

img = cv2.imread('pelota.jpg')
img = cv2.rectangle(img,(100,20),(450,380),(0,255,255),3)
cv2.imshow('rectangulo',img)
```



Dibujar un Circulo:

Para dibujar un círculo, necesita sus coordenadas centrales y su radio. Dibujaremos un círculo dentro del rectángulo dibujado arriba.

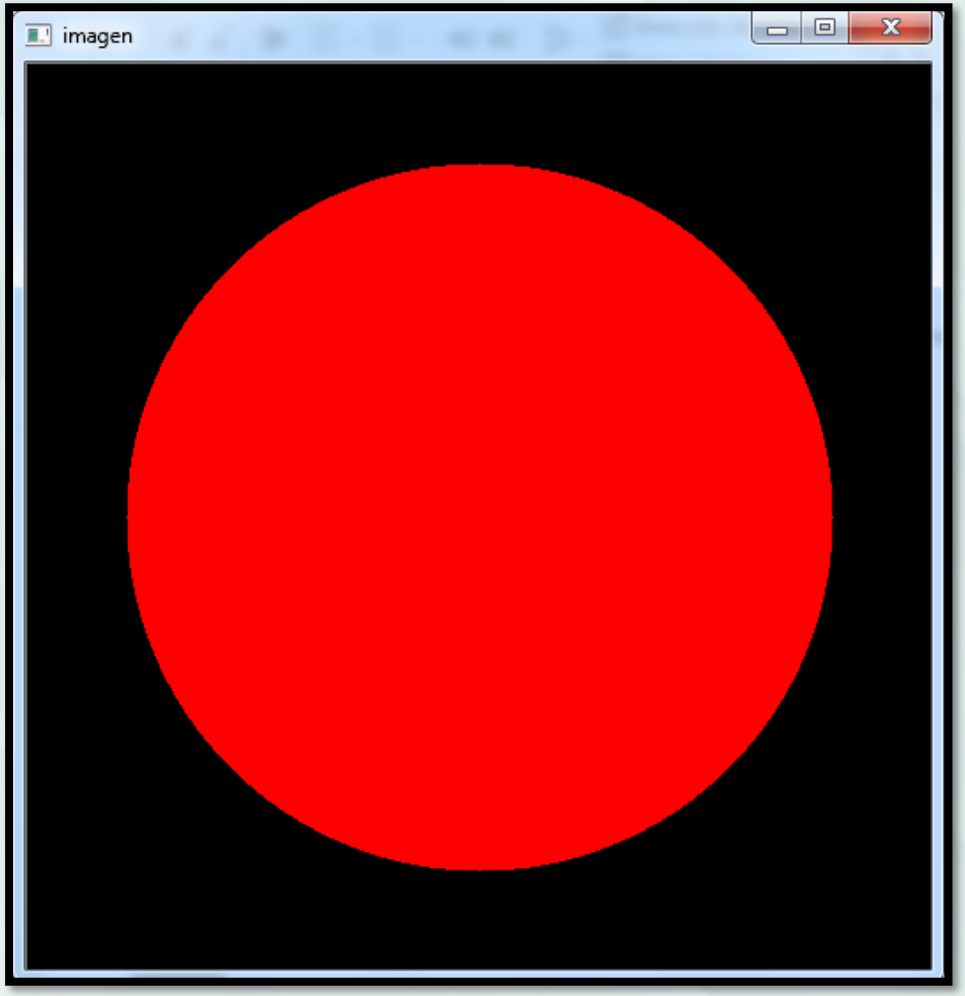
```
import numpy as np
import cv2

img = np.zeros((512,512,3), np.uint8)

img = cv2.circle(img,(256,256), 200, (0,0,255), 5)

cv2.imshow('imagen',img)
```



Dibujar un Circulo:

UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

```
import numpy as np
import cv2

img = np.zeros((512,512,3), np.uint8)

img = cv2.circle(img,(256,256), 200, (0,0,255), -1)

cv2.imshow('imagen',img)
```



Dibujar un Circulo en una Imagen:

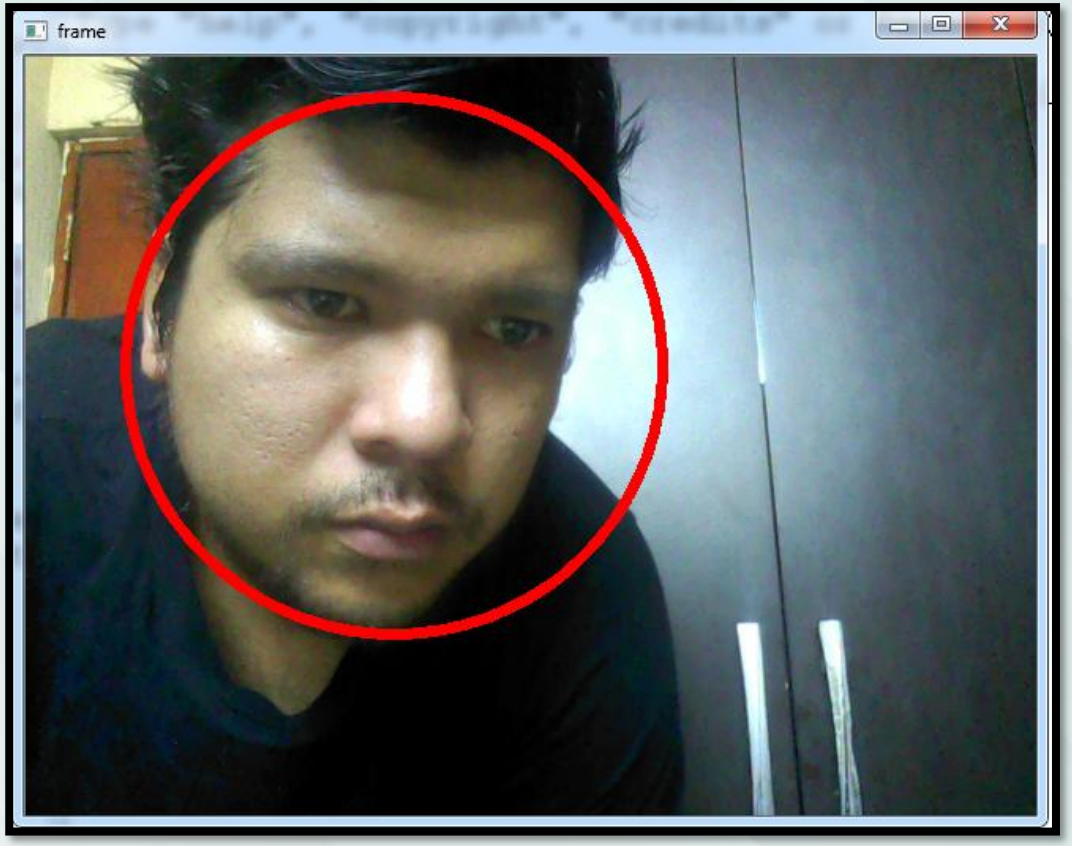
UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

```
import numpy as np
import cv2

img = cv2.imread('pelota.jpg')

img = cv2.circle(img,(233,195), 170, (0,0,255), 5)

cv2.imshow('imagen',img)
```

• Dibujar un Circulo en una Webcam:

UNIVERSIDAD CATÓLICA SEDES SAPIENTIAE

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

while(True):
    ret, frame = cap.read()
    frame = cv2.circle(frame,(233,195), 170, (0,0,255), 5)
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```



UNIVERSIDAD
CATÓLICA
SEDES SAPIENTIAE