CENTRO DE ENSEÑANZA TÉCNICA INDUSTRIAL

INGENIERÍA EN MECATRÓNICA

Visión artificial

Práctica 1:
Cargar una imagen con OpenCv

Alumna:
Vanessa Aguirre Diaz

Registro:
22310274

Fecha:
09 de marzo del 2025

Desarrollo teórico

1.1.5 Operaciones típicas en el procesamiento de imágenes

1. Operaciones Básicas

Lectura y visualización de imágenes

- **Cargar una imagen:** `cv2.imread('imagen.jpg', cv2.IMREAD_COLOR)`
- **Mostrar una imagen:** `cv2.imshow('Ventana', img)`
- **Guardar una imagen:** `cv2.imwrite('nueva_imagen.jpg', img)`

Conversión de formatos de color

- **RGB a escala de grises:** `cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`
- **RGB a HSV:** `cv2.cvtColor(img, cv2.COLOR_BGR2HSV)`

Redimensionamiento y transformación

- **Cambiar tamaño:** `cv2.resize(img, (nuevo_ancho, nueva_altura))`
- **Rotar imagen:**

```
filas, columnas = img.shape[:2]
M = cv2.getRotationMatrix2D((columnas/2, filas/2), 45, 1)
img_rotada = cv2.warpAffine(img, M, (columnas, filas))
```

- **Voltear imagen:**
  - Horizontalmente: `cv2.flip(img, 1)`
  - Verticalmente: `cv2.flip(img, 0)`

2. Mejoras y Filtros

Ajuste de brillo y contraste

- **Aumentar brillo:** `cv2.convertScaleAbs(img, alpha=1.2, beta=50)`
  - `alpha`: factor de contraste
  - `beta`: ajuste de brillo

Suavizado de imágenes (reducción de ruido)

- **Desenfoque Gaussiano:** `cv2.GaussianBlur(img, (5,5), 0)`
- **Desenfoque por promedio:** `cv2.blur(img, (5,5))`
- **Filtro mediano (útil para eliminar ruido de sal y pimienta):** `cv2.medianBlur(img, 5)`

Detección de bordes

- **Filtro Sobel:**

```
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)  #En dirección X
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)  #En dirección Y
```

- **Filtro Canny (muy usado en visión artificial):** `cv2.Canny(img, 100, 200)`

3. Operaciones Morfológicas

Usadas para mejorar o limpiar imágenes binarias.

- **Erosión (reduce ruido):** `cv2.erode(img, kernel, iterations=1)`
- **Dilatación (expande regiones blancas):** `cv2.dilate(img, kernel, iterations=1)`
- **Apertura (erosión + dilatación, elimina ruido):** `cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)`
- **Cierre (dilatación + erosión, rellena huecos):** `cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)`

4. Segmentación y Umbralización

- **Umbralización simple:**

```
_, img_bin = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
```

- **Umbral adaptativo (útil para condiciones de iluminación variables):**

```
img_adapt = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                  cv2.THRESH_BINARY, 11, 2)
```

5. Transformaciones Geométricas

- **Transformación afín (cambio de perspectiva):**

```
pts1 = np.float32([[50,50], [200,50], [50,200]])
pts2 = np.float32([[10,100], [200,50], [100,250]])
M = cv2.getAffineTransform(pts1, pts2)
img_afine = cv2.warpAffine(img, M, (cols, rows))
```

- **Transformación de perspectiva:**

```
M = cv2.getPerspectiveTransform(pts1, pts2)
img_perspectiva = cv2.warpPerspective(img, M, (cols, rows))
```

6. Detección de Características y Objetos

- **Detección de contornos:**

```
contornos, _ = cv2.findContours(img_bin, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img, contornos, -1, (0, 255, 0), 3)
```

- **Detección de esquinas con Harris:**

```
dst = cv2.cornerHarris(img_gray, 2, 3, 0.04)
img[dst > 0.01 * dst.max()] = [0, 0, 255]  # Resalta esquinas en rojo
```

- **Detección de rostros con Haarcascades:**

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(img_gray, 1.3, 5)
for (x,y,w,h) in faces:
    cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
```

1.1.6 Aplicaciones del procesamiento de imágenes digitales

- En el campo de la **medicina y la biomedicina**, el procesamiento de imágenes se utiliza en la obtención y mejora de imágenes médicas como radiografías, tomografías computarizadas (TAC) y resonancias magnéticas. Estas técnicas permiten a los médicos diagnosticar enfermedades con mayor precisión. Además, en la microscopía digital, se emplean algoritmos para analizar células y tejidos, facilitando la detección de anomalías. También se aplica en la genética para el reconocimiento de patrones en secuencias de ADN y proteínas, contribuyendo al diagnóstico de enfermedades hereditarias.
- En **seguridad y vigilancia**, el reconocimiento facial es una herramienta clave para la identificación de personas en aeropuertos, empresas y dispositivos móviles. Además, los sistemas de videovigilancia incorporan algoritmos de detección de movimiento, permitiendo la identificación de actividades sospechosas. Otro ejemplo es el reconocimiento automático de matrículas vehiculares (LPR, por sus siglas en inglés), utilizado para el control de tráfico, peajes y estacionamientos.
- La **industria y la robótica** también se benefician enormemente del procesamiento de imágenes. En manufactura, se emplea para el control de calidad, asegurando que los productos cumplan con los estándares antes de ser distribuidos. Asimismo, en la automatización industrial, los robots utilizan visión artificial para ensamblar piezas y manipular objetos con precisión. En el ámbito logístico, la lectura de códigos de barras y códigos QR permite la gestión eficiente del inventario y la trazabilidad de productos.
- En el sector de **transporte y automoción**, el procesamiento de imágenes es esencial para los sistemas de conducción autónoma. Estos vehículos emplean cámaras y

algoritmos avanzados para detectar señales de tránsito, peatones y otros vehículos, garantizando una conducción segura. Además, se han desarrollado sistemas de detección de fatiga en conductores, analizando la expresión facial y el movimiento ocular para prevenir accidentes. En el ámbito del tráfico, el análisis de imágenes permite el monitoreo en tiempo real de la circulación vehicular, optimizando la sincronización de semáforos y reduciendo congestiones.

- Dentro de la **inteligencia artificial y el aprendizaje automático**, el procesamiento de imágenes es clave en el reconocimiento de objetos y escenas. Las redes neuronales convolucionales (CNN) permiten clasificar imágenes con una precisión sorprendente, lo que se aplica en motores de búsqueda, redes sociales y sistemas de recomendación. En la realidad aumentada y la realidad virtual, las cámaras y sensores detectan el entorno y los movimientos del usuario para generar experiencias interactivas. Asimismo, las aplicaciones de redes sociales utilizan algoritmos para agregar filtros y efectos a las fotos y videos en tiempo real.
- En el ámbito de la **astronomía y la exploración espacial**, el procesamiento de imágenes juega un papel fundamental en el análisis de imágenes satelitales y telescópicas. Los astrónomos utilizan algoritmos para mejorar la calidad de las imágenes capturadas por telescopios espaciales y detectar planetas, estrellas y galaxias distantes. Además, se aplican técnicas de reducción de ruido y mejora del contraste para interpretar mejor los datos obtenidos del espacio exterior.
- La **agricultura y el medio ambiente** también se ven beneficiados por el uso de visión artificial. En la agricultura de precisión, los drones equipados con cámaras analizan los cultivos para detectar plagas, enfermedades y zonas con deficiencias de riego. En el monitoreo ambiental, el procesamiento de imágenes satelitales permite el estudio del cambio climático, la deforestación y el derretimiento de los glaciares. También se emplea en sistemas de clasificación de residuos para separar automáticamente materiales reciclables, contribuyendo a la gestión sostenible de los desechos.
- En el mundo del **entretenimiento y la multimedia**, las técnicas de procesamiento de imágenes se aplican en la edición y restauración de fotografías y videos. Se pueden mejorar imágenes antiguas, eliminar ruido y aumentar la resolución mediante algoritmos avanzados. En la compresión de imágenes y videos, formatos como JPEG y MPEG permiten reducir el tamaño de los archivos sin comprometer significativamente la calidad. Asimismo, en la industria cinematográfica y de los videojuegos, los efectos especiales generados por computadora (CGI) utilizan procesamiento de imágenes para crear escenarios y personajes hiperrealistas.

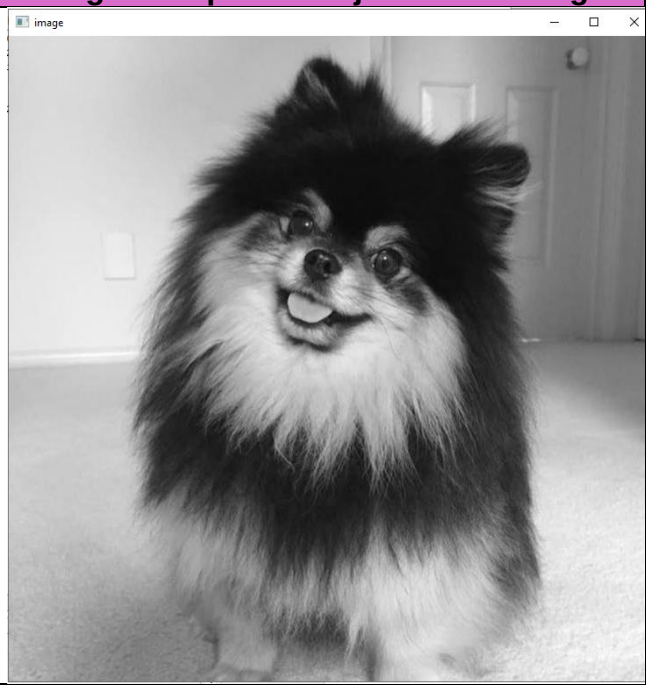
Desarrollo práctico:

Parte 1: Cargar una imagen con OpenCv

Código:

```
# Practica 1 Pt.1: Cargar una imagen

# Por Vanessa Aguirre Diaz



import cv2

# Importa la biblioteca OpenCV, que se usa para procesamiento de imágenes y
visión artificial.

import numpy as np

# Importa NumPy, que es útil para manejar matrices y operaciones
matemáticas.

from matplotlib import pyplot as plt

# Importa pyplot de Matplotlib, aunque en este código no se usa.

img = cv2.imread('perro.jpg',cv2.IMREAD_GRAYSCALE)

# Carga la imagen llamada perro.jpg en modo escala de grises
(cv2.IMREAD_GRAYSCALE equivale a 0).

cv2.imshow('image',img)

# Muestra la imagen en una ventana con el título image.

cv2.waitKey(0)

# Espera a que el usuario presione una tecla para cerrar la ventana.

cv2.destroyAllWindows()

# Cierra todas las ventanas de OpenCV abiertas.
```

Resultados:

Imagen original	Imagen después de ejecutar el código
	

Parte 2: cargar una imagen, convertirla a blanco y negro y dibujar con líneas sobre ella.

Código:

```
# Practica 1 Pt.2: Cargar una imagen en escala de grises y dibujar varias lineas de color azul

# Por Vanessa Aguirre Diaz

# Importamos las bibliotecas necesarias:

import cv2

# OpenCV para procesamiento de imágenes

import numpy as np

# NumPy (no se usa en este código, pero está importado)

from matplotlib import pyplot as plt

# Matplotlib para visualizar la imagen
```

```
img = cv2.imread('perro.jpg', cv2.IMREAD_GRAYSCALE)

# Cargamos la imagen 'perro.jpg' en escala de grises

# Mostramos la imagen con Matplotlib

plt.imshow(img, cmap='gray', interpolation='bicubic')   # cmap='gray' la muestra en escala de grises

# Usa interpolación bicubic para mejorar la visualización.

plt.xticks([]), plt.yticks([])

# Ocultamos los valores de los ejes X e Y para una visualización más limpia

# Dibujamos una línea en color azul ('b') con un grosor de 5 píxeles

plt.plot([200, 200, 400, 600, 600, 500, 400, 300, 200], [100, 300, 500, 300, 100, 50, 150, 50, 100], 'b', linewidth=5)    #x,y

plt.show() # Mostramos la imagen con la línea dibujada encima
```

Resultados:

Imagen original	Imagen después de aplicar el código