Introducción a las OpenCV

Cátedra Visión por Computadoras

8 de abril de 2025

OpenCV

Historia de las OpenCV

- Nacen en 1999 en Intel
- Luego continúa su desarrollo en WillowGarage
- En la actualidad es mantenida por la comunidad y cualquier persona puede aportar código a la librería
- Tiene licencia de código abierto BSD

Características de las OpenCV

OpenCV

- Librería para procesamiento de imágenes y visión por computadora
- Escrita en C++, pero con interfaz para:
  - **Python**
  - Java
  - MATLAB/OCTAVE
  - C#
  - Perl
  - Ch
  - Haskell
  - Ruby
- Soporta múltiples sistemas operativos:
  - **Linux**
  - Windows
  - macOS
  - FreeBSD
  - NetBSD
  - OpenBSD
  - Android

Características de las OpenCV

Aceleración de Hardware

- Soporta IPP (Intel Performance Primitives)
- CUDA
- OpenCL (Vulkan, todavia no del todo integrada)

Características de las OpenCV

Módulos Principales

- core. Core Functionality
- imgproc. Image processing
- imgcodecs. Image file reading and writing
- videoio. Media I/O
- highgui. High-level GUI
- video. Video Analysis
- calib3d. Camera Calibration and 3D Reconstruction
- features2d. 2D Features Framework
- objdetect. Object Detection
- ml. Machine Learning
- flann. Clustering and Search in Multi-Dimensional Spaces
- photo. Computational Photography
- stitching. Images stitching

Instalación en Linux

Debian

- Podemos instalar OpenCV desde el repositorio oficial de Debian
- Si queremos la última versión, podemos bajar el código fuente y compilarlo

Instalación desde el repositorio oficial

```
sudo apt update && sudo apt install libopencv-dev python3-opencv
```

OpenCV-Python

Algunos comentarios . . .

- Sabemos que C/C++ es más potente computacionalmente que Python, pero también más difícil de codificar, mantener y reutilizar.
- OpenCV ofrece código computacionalmente intensivo escrito en C++, que puede ser usado como un módulo de Python a través de un adaptador (wrapper).
- De esta manera nuestro código mixto es casi tan rápido como el código en C++, pero más simple y rápido para codificar.
- Además OpenCV usa *numpy arrays*, que son arreglos muy optimizados de una librería de Python para cálculo numérico.

Empezando con imágenes

Como leer una imagen

- Para esto se usa la función **`cv2.imread(filename[, flags])`**
- **filename** es el nombre relativo del archivo o el camino completo
- Las **flags** son opcionales, pero la más usada es la que indica el modo de lectura:
  - **`cv2.IMREAD_COLOR`** Abre la imagen a color (Si tiene transparencias se descarta) Es la opción por defecto.
  - **`cv2.IMREAD_GRAYSCALE`** Abre la imagen en escala de grises.
  - **`cv2.IMREAD_UNCHANGED`** Abre la imagen incluyendo el canal alpha.

Ejemplo

```
import cv2

# Lee una imagen en escala de grises
img = cv2.imread('messi5.jpg', cv2.IMREAD_GRAYSCALE)
```

Empezando con imágenes

Como mostrar una imagen

- Usamos la función **`cv2.imshow(ventana, img)`**
- **`ventana`** es el nombre de la ventana, una cadena
- **`img`** es la imagen que vamos a mostrar
- **`cv2.waitKey([, retardo])`**
  - Espera durante **`retardo`** ms a que se presione una tecla
  - Si **`retardo = 0`** espera indefinidamente
- **`cv2.destroyAllWindows()`** destruye todas las ventanas
- **`cv2.destroyWindow(ventana)`** destruye la ventana con nombre **`ventana`**

Ejemplo

```
...
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Empezando con imágenes

Escribiendo una imagen

- **`cv2.imwrite(filename, img[, params])`**
  - **filename** es el nombre relativo del archivo o el camino completo en donde queremos guardar la imagen
  - **img** es la imagen que queremos guardar
  - Los **params** son opcionales y dependen del tipo de imagen a guardar, por ejemplo **`png`**, **`jpg`**

Ejemplo

```
...
cv2.imwrite('messigris.png', img)
```

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2

img = cv2.imread('homero.jpg', cv2.IMREAD_GRAYSCALE)
cv2.imshow('Imagen', img)
k = cv2.waitKey(0)

if k == ord('g'):    #si se presiona la letra 'g' se guarda la imagen
    print('Guardando imagen en escala de grises.')
    cv2.imwrite('homero_gris.png', img)
else:
    print('Imagen no guardada.')

cv2.destroyAllWindows()
```

Empezando con videos

Como capturar un video

- Para esto se usa la clase **`cv2.VideoCapture(device)`**
  - **`device`** indica el número o nombre del dispositivo, ej. 0 o **`'/dev/video0'`**,
  - Si tenemos más de una cámara usamos 0, 1, 2, . . . o **`'/dev/video0'`**, **`'/dev/video1'`**, **`'/dev/video2'`**
  - **`device`** también puede ser una cadena que indique el nombre de archivo
- Nos devuelve un objeto que representa a la cámara
- Para verificar que esté correctamente abierto usamos el método **`isOpened()`**
- Para capturar una imagen usamos **`read()`**
- Para cerrar la cámara usamos **`release()`**

Empezando con videos - Como capturar un video

Ejemplo

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

cap = cv2.VideoCapture(0)

while(True):
    # devuelve una tupla, en ret el estado en img la imagen
    ret, img = cap.read()

    # alguna operación sobre img...
    gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    cv2.imshow('img', gris)
    # esta and con FF es para considerar sólo los primeros 8 bits
    if((cv2.waitKey(1) & 0xFF) == ord('q')):
        break

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

Empezando con videos - Abriendo un video desde un archivo

Ejemplo

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import cv2

if(len(sys.argv) > 1):
    mivideo = sys.argv[1]
else:
    print('Pasar el nombre del video como argumento')
    sys.exit(0)

cap = cv2.VideoCapture(mivideo)

while(cap.isOpened()):
    ret, img = cap.read()

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    cv2.imshow('Mi Video', gray)
    if((cv2.waitKey(33) & 0xFF) == ord('q')):
        break

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

Práctica

¿Cómo obtener la cantidad de imágenes por segundo (o fps) usando las OpenCV?

Usarlo para no tener que *harcodear* el **`retardo`** de la función **`waitKey`**.

Empezando con videos

Como guardar un video

- Usamos la clase:
  `cv2.VideoWriter([filename, fourcc, fps, frameSize])`
  - **`filename`** es el nombre del archivo de salida
  - **`fourcc`** es la codificación del video, DIVX, XVID, MJPG, X264, WMV1, WMV2.
  - **`fps`** es el número de cuadros por segundo (frames per second)
  - **`frameSize`** tupla con la resolución del video a guardar, (W, H) = (ancho, alto)
- El formato lo especificamos con **`cv2.VideoWriter_fourcc`**, esto se usa para generar un código con cuatro carácteres (four character code)

Empezando con videos - Como guardar un video

Ejemplo

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

fps = 25
anchoalto = (1280,720)
# Configuración para MP4 con H.264 (requiere FFmpeg)
fourcc = cv2.VideoWriter_fourcc(*'H264')
out = cv2.VideoWriter('mivideo.mp4', fourcc, fps, anchoalto)
#sino probar 'MJPG' -> .avi

# Capturar video de cámara
cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    out.write(frame)
    cv2.imshow('Video', frame)
    if cv2.waitKey(1) == ord('q'):
        break

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

Práctica, continuación

¿Cómo obtener el ancho y alto de las imágenes capturadas usando las OpenCV?

Usarlo para no tener que *harcodear* el **`anchoalto`** del video generado.

Dibujando líneas

- `img = cv.line(img, pt1, pt2, color[, thickness[, lineType]])`
- **`img`** imagen sobre la que vamos a dibujar, es la misma que la de salida
- **`pt1`** punto inicial de la línea, indicado con una tupla **`(x, y)`**
- **`pt2`** punto final de la línea, indicado con una tupla **`(x, y)`**
- **`color`** color de la línea, indicado con una tupla **`(B, G, R)`**
- **`thickness`** ancho de la línea en pixeles, entero
- **`lineType`** tipo de línea
  - **`cv.LINE_8`**
  - **`cv.LINE_4`**
  - **`cv.LINE_AA`**

Dibujando rectángulos y círculos

Rectángulo

- `img = cv.rectangle(img, pt1, pt2, color[, thickness[, lineType]])`

Círculo

- `img = cv.circle(img, center, radius, color[, thickness[, lineType]])`
- **`center`** centro del círculo, indicado con una tupla **`(x, y)`**
- **`radius`** radio del círculo en pixeles, entero

Dibujando elipses

Elipse a la una...

- `img = cv.ellipse(img, center, axes, angle, startAngle, endAngle, color[, thickness[, lineType]])`
- **`center`** centro del círculo, indicado con una tupla `(x, y)`
- **`axes`** largo de los ejes mayor y menor, indicado con una tupla `(w, h)`
- **`angle`** ángulo de inclinación del eje mayor
- **`startAngle`** ángulo de inicio
- **`endAngle`** ángulo de finalización

Elipse a las dos

- `img = cv.ellipse(img, box, color[, thickness[, lineType]])`
- **`box`** caja contenedora de la elipse

Dibujando polígonos

Polígonos

- **`img = cv.polylines(img, pts, isClosed, color[, thickness[, lineType]])`**
- **`pts`** arreglo con los puntos vértices de las líneas, debe ser un arreplo de dimensiones **`(npuntos, 1, 2)`**, **`[[[x0, y0]], [[x1, y1]], ...]`**
- **`isClosed`** bool que indica si hay que unir el último punto con el primero

Agregando texto

Texto

- `img = cv.putText(img, text, org, fontFace, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])`
- **`text`** texto a escribir
- **`org`** ubicación del punto inferior izquierdo de la imagen especificado como una tupla
- **`fontFace`** fuente
- **`fontScale`** factor para escalar la fuente, float
- **`bottomLeftOrigin`** bool que indica tomar como inicio de la imagen el borde inferior izquierdo

Ejemplo

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
import cv2 as cv

azul = (255, 0, 0); verde = (0, 255, 0); rojo = (0, 0, 255)

# Creamos una imagen RGB de color negro
img = np.zeros((512, 512, 3), np.uint8)

cv.line(img, (100, 100), (200, 100), azul, 5)
cv.imshow('frame', img); key = cv.waitKey(0)
cv.rectangle(img, (284, 50), (410, 178), verde, 1)
cv.imshow('frame', img); key = cv.waitKey(0)
cv.circle(img, (347, 113), 63, rojo, -1)
cv.imshow('frame', img); key = cv.waitKey(0)
cv.ellipse(img, (256, 256), (100, 150), 0, 0, 180, rojo, -1)
cv.imshow('frame', img); key = cv.waitKey(0)
pts = np.array([[90, 45], [215, 5], [225, 55], [205, 35]], np.int32)
pts = pts.reshape((-1, 1, 2))
cv.polylines(img, [pts], True, (0, 255, 255))
pts = np.array([[422, 45], [297, 5], [287, 55], [307, 35]], np.int32)
pts = pts.reshape((-1, 1, 2))
cv.polylines(img, [pts], True, (0, 255, 255))
cv.imshow('frame', img); key = cv.waitKey(0)
font = cv.FONT_HERSHEY_SIMPLEX
cv.putText(img, 'CV2025', (10, 500), font, 4, (255, 255, 255), 2, cv.LINE_AA)

cv.imshow('frame', img)
key = cv.waitKey(0)
cv.destroyAllWindows()
```

Eventos del mouse

Un evento es una acción realizada por el usuario (ejemplo presionar algún botón del mouse), que puede ser capturada para actuar en consecuencia (dibujar un punto en la imagen).

Algunos eventos...

EVENT_MOUSEMOVE	el puntero del mouse se movió
EVENT_LBUTTONDOWN	se apretó el botón izquierdo
EVENT_RBUTTONDOWN	se apretó el botón derecho
EVENT_MBUTTONDOWN	se apretó el botón del medio
EVENT_LBUTTONUP	se soltó el botón izquierdo
EVENT_RBUTTONUP	se soltó el botón derecho
EVENT_MBUTTONUP	se soltó el botón del medio
EVENT_LBUTTONDBLCLK	doble click con el botón izquierdo
EVENT_RBUTTONDBLCLK	doble click con el botón derecho
EVENT_MBUTTONDBLCLK	doble click con el botón del medio
EVENT_MOUSEWHEEL	scroll vertical, positivo significa hacia adelante y negativo hacia atrás
EVENT_MOUSEHWHEEL	scroll horizontal, positivo significa hacia la derecha y negativo hacia la izquierda

Dibujando puntos y líneas

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2
import numpy as np

azul = (255, 0, 0); verde = (0, 255, 0); rojo = (0, 0, 255)
xybutton_down= 0,0
# mouse callback
def dibuja(event, x, y, flags, param):
    global xybutton_down
    if event == cv2.EVENT_LBUTTONDOWN:
        print("cv2.EVENT_LBUTTONDOWN", event)
        xybutton_down = x,y
        cv2.circle(img, xybutton_down, 9, rojo, 2)
    elif event == cv2.EVENT_RBUTTONDOWN:
        print("cv2.EVENT_RBUTTONDOWN", event)
        cv2.circle(img, (x, y), 5, verde, -1)
    elif event == cv2.EVENT_LBUTTONUP:
        print("cv2.EVENT_LBUTTONUP", event)
        cv2.line(img, xybutton_down, (x, y), azul, 3)

img = np.ones((600, 800, 3), np.uint8) * 100
# Creamos una una ventana y capturamos los eventos del mouse en esa ventana
cv2.namedWindow('imagen')
cv2.setMouseCallback('imagen', dibuja)

while(1):
    # usamos la ventana creada para mostrar la imagen
    cv2.imshow('imagen', img)
    if cv2.waitKey(20) & 0xFF == 27: # ESC
        break
cv2.destroyAllWindows()
```

Dibujando rectángulos y líneas

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2
import numpy as np

azul = (255, 0, 0); verde = (0, 255, 0); rojo = (0, 0, 255)
dibujando = False    # True si el botón está presionado
modo = True    # si True, rectángulo, sino línea, cambia con 'm'
xybutton_down = -1, -1

def dibuja(event, x, y, flags, param):
    global xybutton_down, dibujando, modo
    if event == cv2.EVENT_LBUTTONDOWN:
        dibujando = True
        xybutton_down = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if dibujando is True:
            img[:] = 0
            if modo is True:
                cv2.rectangle(img, xybutton_down, (x, y), azul, -1)
            else:
                cv2.line(img, xybutton_down, (x, y), rojo, 2)
    elif event == cv2.EVENT_LBUTTONUP:
        dibujando = False

img = np.zeros((512, 512, 3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image', dibuja)
while(1):
    cv2.imshow('image', img)
    k = cv2.waitKey(1) & 0xFF
    if k == ord('m'):
        modo = not modo
    elif k == 27:
        break
cv2.destroyAllWindows()
```

Práctica - Manipulación de imágenes

- Usando como base el programa que se muestra en la siguiente página, escribir un programa que permita seleccionar una porción rectangular de una imagen, luego
  - con la letra “g” guardar la porción de la imagen seleccionada como una nueva imagen,
  - con la letra “r” restaurar la imagen original y permitir realizar una nueva selección,
  - con la “q” finalizar.

```python
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2
import numpy as np

dibujando = False    # true if mouse is pressed
modo = True    # if True, draw rectangle. Press 'm' to toggle to curve
ix, iy = -1, -1

def draw_circle(event, x, y, flags, param):
    global ix, iy, dibujando, modo
    if event == cv2.EVENT_LBUTTONDOWN:
        dibujando = True
        ix, iy = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if dibujando is True:
            if modo is True:
                cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), -1)
            else:
                cv2.circle(img, (x, y), 5, (0, 0, 255), -1)
    elif event == cv2.EVENT_LBUTTONUP:
        dibujando = False
        if modo is True:
            cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), -1)
        else:
            cv2.circle(img, (x, y), 5, (0, 0, 255), -1)

img = np.zeros((512, 512, 3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_circle)
while(1):
    cv2.imshow('image', img)
    k = cv2.waitKey(1) & 0xFF
    if k == ord('m'):
        modo = not modo
    elif k == 27:
        break
cv2.destroyAllWindows()
```