Práctica de visión computacional o visión artificial

Objetivo

Mediante el uso de visión computacional reconocer una imagen dentro de otra imagen y un objeto dentro de un video utilizando lenguaje de programación Python y su librería OpenCV.

Fundamento teórico

La visión computacional es el estudio de los procesos de la visión de los organismos vivos, que utilizan para identificar y ubicar objetos a través del procesamiento de imágenes para entenderlos y desarrollar máquinas que tengan incorporadas esas capacidades (Sucar & Gómez, s.f.).

Marr, 77 (citado por Sucar & Gómez s.f.) define “Visión es un proceso que produce a partir de las imágenes del mundo exterior una descripción que es útil para el observador y que no tiene información irrelevante”.

El objetivo de la visión computacional es obtener propiedades de una imagen para que la computadora pueda describirlas e interpretarlas.

IBM (s.f.) presenta la siguiente definición para visión artificial:

> La visión artificial es un campo de la IA que permite que las computadoras y los sistemas obtengan información significativa de imágenes digitales, videos y otras entradas visuales, y tomen acciones o hagan recomendaciones basadas en esa información. Si la IA permite que las computadoras piensen, la visión artificial les permite ver, observar y comprender.
>
> La visión artificial funciona de manera muy similar a la visión humana, excepto que los humanos tienen una ventaja. La vista humana tiene la ventaja de las experiencias y los contextos aprendidos para diferenciar entre los objetos, qué tan lejos están, si se están moviendo o si hay algo mal en una imagen.

Russula (2021) explica que la visión de máquina es una subcategoría de la visión artificial “La visión de máquina utiliza un procesamiento basado en controladores integrados, construido para analizar rápidamente los datos y tomar decisiones sencillas y automatizadas, para el control de calidad, la inspección y el guiado”.

La visión de máquina utiliza un procesamiento basado en controladores integrados, construido para analizar rápidamente los datos y tomar decisiones sencillas y automatizadas, para el control de calidad, la inspección y el guiado.

Para Sucar & Gómez (s.f.) el proceso de visión computacional se puede dividir en 3 grandes etapas:

- Procesamiento de nivel bajo - se trabaja directamente con las imágenes para extraer propiedades como orillas, gradiente, profundidad, textura, color, etc.
- Procesamiento de nivel intermedio - consiste generalmente en agrupar los elementos obtenidos en el nivel bajo, para obtener, por ejemplo, contornos y regiones, generalmente con el propósito de segmentación.
- Procesamiento de alto nivel - consiste en la interpretación de los entes obtenidos en los niveles inferiores y se utilizan modelos y/o conocimiento a priori del dominio.

IBM (s. f.) menciona como tareas de la visión artificial: clasificación de imágenes y videos, detección de objetos, seguimiento de objetos, recuperación de imágenes basada en contenido.

OpenCV

OpenCV (Open Source Computer Vision Library) es una librería libre desarrollada por Intel en 1999. Inicialmente fue desarrollada en C/C++, es multiplataforma, corre en diferentes sistemas operativos como Linux, Windows, Mac OS X, Android e iOS. Además, se puede usar en diferentes lenguajes de programación como Java, Objective C, Python y C# (Del Valle, s.f.).

Requerimientos

Lenguaje Phyton
IDE de Python
Librería OpenCV
Librería NumPy

2 imágenes

1 video

Práctica 1

Actividad 1

En esta parte utilizaremos la librería OpenCV para reconocer una imagen que se encuentra ubicada en otra imagen. Ambas imágenes deberán estar en la misma carpeta. A continuación, se presenta el código que deberá programarse en lenguaje Python y se utilizarán además las librerías OpenCV y NumPy, las cuales deberán instalarse previamente, escribiendo lo siguiente en la línea de órdenes (CMD) del sistema operativo en modo administrador:

pip3 install opencv-python

pip3 install numpy

```
# Práctica 1 Buscar un objeto en una imagen
#Importamos las librerías
import cv2
import numpy as np

"""Mostrar las versiones de las librerías para ver que todo bien"""
print('versión: ', cv2.__version__)
print('versión: ', np.__version__)

"""Se importan las imágenes de la placa y el circuito a reconocer, se guardan en variables correspondientes, se importa en escala de grises, se especifica un segundo parámetro. Toda la información se guarda en matrices."""
placa = cv2.imread('placa.jpg', cv2.IMREAD_GRAYSCALE)
circuito = cv2.imread('circuito.jpg', cv2.IMREAD_GRAYSCALE)

"""Se definen las dimensiones del circuito con la función shape, se imprimen"""
alto, ancho =np.shape(circuito)
```

```
print(alto , ancho)

"""Se aplica función matemática a plantillas para buscar semejanzas por medio del método coeficiente normalizado en la matriz resultado"""

resultado = cv2.matchTemplate(placa, circuito, cv2.TM_CCOEFF_NORMED)
min, max, pos_min, pos_max = cv2.minMaxLoc(resultado)
print(min, max, pos_min, pos_max)

# Se identifican los pixeles superior izquierda e inferior derecha
pixel_sup_izq = pos_max
pixel_inf_der = (pos_max[0] + ancho, pos_max[1] + alto)

#Se genera el rectángulo de color que identificará el circuito en la placa
placa= cv2.imread('placa.jpg')
color = (0, 255, 255) #BGR
cv2.rectangle(placa, pixel_sup_izq, pixel_inf_der, color, 4)

“”” Se busca la imagen, se presenta el resultado del reconocimiento, espera una tecla para terminar y destruye todo.”””””
cv2.imshow('Practica1', placa)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Actividad 2

Buscar 2 nuevas imágenes y repetir el proceso. Modificando el código.

Práctica 2

Ahora vamos a realizar el procesamiento de un video para reconocer un objeto de cierto color.

Para ello deberán programar el siguiente código:

Práctica 2

Actividad 1

```
# Práctica 2 Seguimiento de un objeto en un video
# Como se ejecuta
# python practica2.py --video seguimiento.mp4

#Importamos paquetes y librerías necesarios
from collections import deque
from imutils.video import VideoStream

import numpy as np
import argparse
import cv2
import imutils
import time

#Se construye el argumento que se va a analizar y se analizan los argumentos
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="activar la ruta de ubicación del video")
ap.add_argument("-b", "--buffer", type = int, default=16, help="Tamaño máximo del
buffer")

args= vars(ap.parse_args())

""" Se definen los límites inferior y superior de la pelota "verde"
en el espacio de color HSV (Hue, Saturation, Brihghtness), luego inicializa la lista
de puntos rastreados"""

greenLower = (29, 86, 6) #BGR
greenUpper = (64, 255, 100) #BGR
```

```
pts = deque(maxlen = args["buffer"])

"""Si no se proporciona ruta del video, busca la cámara web"""
if not args.get("video", False):
    vs = VideoStream(scr=0).start()

# En otro caso, toma como referencia el archivo de video
else:
    vs = cv2.VideoCapture(args["video"])

# Permite el archivo de video o la cámara sea presentado
time.sleep(2.0)
# continua el ciclo
while True:
    #Toma el cuadro actual
      frame = vs.read()
      # Manejar el marco VideoCapture o VideoStream
      frame = frame[1] if args.get("video", False) else frame
"""si estamos viendo un video y no tomamos un cuadro,
entonces hemos llegado al final del video"""
      if frame is None:
        break
      """Cambiar el tamaño del marco, desenfocar y convertir al espacio de color HSV"""
      frame = imutils.resize(frame, width = 600)
      blurred = cv2.GaussianBlur(frame, (11, 11), 0)
      hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
      """Construir una máscara para el color "verde", luego se realizan
      una serie de dilataciones y erosiones para eliminar cualquier pequeña mancha en la máscara"""
      mask = cv2.inRange(hsv, greenLower, greenUpper)
```

```
        mask = cv2.erode(mask, None, iterations = 2)
        mask = cv2.dilate(mask, None, iterations = 2)
"""Encontrar contornos en la mascara e inicializar el centro actual de la imagen (x,y)"""
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        center = None
#Proceder solo si se ha encontrado al menos un contorno
        if len(cnts) > 0:
            """Encontrar el contorno más grande en la máscara, luego calcular el
            círculo envolvente mínimo y centroide"""
              c = max(cnts, key =cv2.contourArea)
              ((x,y), radius) = cv2.minEnclosingCircle(c)
              M = cv2.moments(c)
              center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
              """Proceder solo si el radio cumple con un tamaño mínimo"""
              if radius > 10:
                  """dibujar el círculo y el centroide en el marco,
                    luego actualizar la lista de puntos rastreados"""
                      cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
                      cv2.circle(frame, center, 5, (0,0,255), -1)
        #Actualizar la cola de puntos
        pts.appendleft(center)
        #Bucle sobre el conjunto de puntos rastreados
        for i in range(1, len(pts)):


            #Si no es ninguno de los puntos rastreados ignorarlos
                if pts[i -1] is None or pts[i] is None:
                 continue
```

```
            """En otro caso, calcular el grosor de la línea y
            dibujar las líneas de conexión"""
            thickness = int(np.sqrt(args["buffer"] / float(i + 1)) * 2.5)
            cv2.line(frame, pts[i -1], pts[i], (0, 0, 255), thickness)

     #Mostrar el marco en la pantalla
    cv2.imshow("Practica 2", frame)
    key = cv2.waitKey(1) & 0xFF
     #Detener el ciclo si se presiona la tecla q
    if key == ord("q"):
        break
#Si no se utiliza un archivo de video, detener la transmisión de video de la cámara
if not args.get("video", False):
            vs.stop()
#En otro caso liberar la cámara
else:
   vs.release()
   #Cerrar todas las ventanas
cv2.destroyAllWindows()
```

Actividad 2

Agregar un video diferente y reconocer un objeto de diferente color al de la actividad anterior.

Resultados

Conclusiones

Referencias

Del Valle, L. (s.f.). 81. Visión artificial, OpenCV y Python. Programarfacil. https://programarfacil.com/podcast/81-vision-artificial-opencv-phyton/

IBM. (s. f.). ¿Qué es la visión Artificial?

https://www.ibm.com/mx-es/topics/computer-vision

Sucar, E. & Gómez, G. (s.f.). Visión computacional. INAOE.
https://ccc.inaoep.mx/~esucar/Libros/vision-sucar-gomez.pdf

.