



## ***Visión Artificial***

***No. de Practica: 2***

***Título:*** Loading Video Source, Image Operations and Image arithmetics and Logic OpenCV Python Tutorial

***Nombre:*** Aldo Misael Osuna Rodríguez

***Registro:*** 22310221

***6°G***

***19-marzo-2025***

## Objetivo:

En este tutorial de OpenCV con Python, cubriremos algunas operaciones básicas con vídeo y cámaras web.

## Código:

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))

while(True):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    out.write(frame)
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

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

## Comentarios:

Primero importamos "numpy" and "cv2" y despues en nuestro codigo de python cambiamos en cv2.VideoCapture(1), de 1 a 0 dentro del parentesis de VideoCapture(0). Esto es porque al tener dentro de VideoCapture() un numero 1 estamos mandando a llamar a una camara secundaria/externa, entonces al cambiar este 1 por un 0 estamos activando a nuestra camara principa que es la que esta integrada en nuestro dispositivo.

Este código inicia un bucle infinito (que se romperá posteriormente con una sentencia break), donde ret y frame se definen como cap.read(). Básicamente, ret es un booleano que indica si hubo un retorno, y frame es cada fotograma devuelto. Si no hay ningún fotograma, no se generará un error, sino None.

```
while(True):
    ret, frame = cap.read()
```

Aquí, definimos una nueva variable, gris, como el marco, convertido a gris. Observe que indica BGR2GRAY. Es importante tener en cuenta que OpenCV interpreta los colores como BGR (Azul, Verde, Rojo), mientras que la mayoría de las aplicaciones informáticas los interpretan como RGB (Rojo, Verde, Azul).

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Tenga en cuenta que, a pesar de ser una transmisión de video, seguimos usando imshow. Aquí, mostramos la señal convertida a gris. Si desea mostrar ambas simultáneamente, puede usar imshow para el fotograma original e imshow para el gris, y así aparecerán dos ventanas.

```
cv2.imshow('frame',gray)
```

Esta sentencia solo se ejecuta una vez por fotograma. Básicamente, si obtenemos una clave, y esa clave es una q, saldremos del bucle while con un break, que luego se ejecuta:

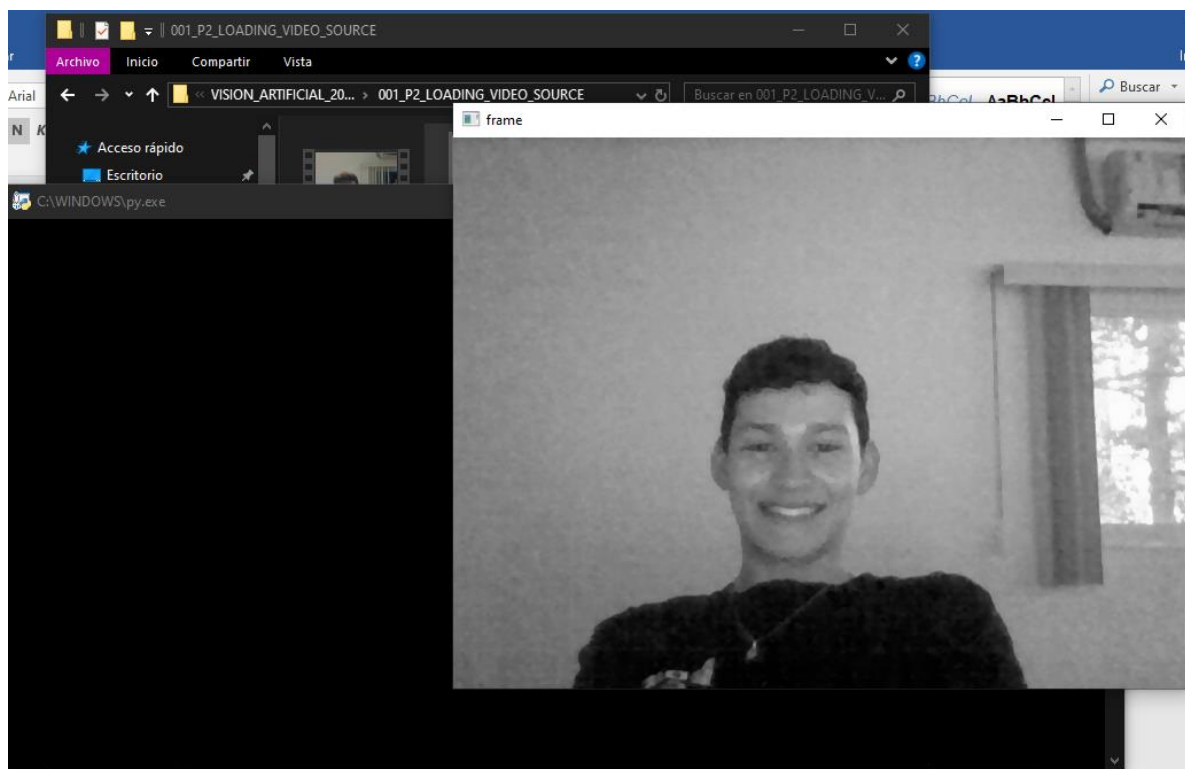
```
if cv2.waitKey(1) & 0xFF == ord('q'):  
    break
```

Esto libera la cámara web y cierra todas las ventanas de imshow().

En algunos casos, es posible que quieras grabar y guardar la grabación en un nuevo archivo.

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

## Resultados:



## Objetivo:

Todo vídeo se divide en fotogramas. Cada fotograma, al igual que una imagen, se divide en píxeles almacenados en filas y columnas dentro del fotograma/imagen. Cada píxel tiene una ubicación de coordenadas y se compone de valores de color. Veamos algunos ejemplos de cómo acceder a diversos aspectos de estos principios.

## Código:

```
import cv2
import numpy as np

img = cv2.imread('watch2.jpg',cv2.IMREAD_COLOR)

img[55,55] = [255,255,255]
px = img[55,55]
print(px)

img[100:150,100:150] = [255,255,255]
px = img[100:150,100:150]
print(px)

print(img.shape)
print(img.size)
print(img.dtype)

watch_face = img[37:111,107:194]
img[0:74,0:87] = watch_face

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

## Comentarios:

Primero iniciamos cargando nuestra imagen que en este caso esta guardada en el dispositivo como "watch2.jpg", después podemos referenciar pixeles específicos:

```
px = img[55,55]
```

Después podemos cambiar un pixel

```
img[55,55] = [255,255,255]
```

y volver a referenciar

```
px = img[55,55]
```

```
print(px)
```

Esto modifica nuestro pixel por lo que debería ser diferente ahora. Después podemos referenciar la región de la imagen:

```
px = img[100:150,100:150]
```

```
print(px)
```

y podemos modificar la región de la imagen (ROI):

```
img[100:150,100:150] = [255,255,255]
```

Podemos referenciar ciertas características de nuestra imagen:

```
print(img.shape)
```

```
print(img.size)
```

```
print(img.dtype)
```

y podemos realizar operaciones como:

```
watch_face = img[37:111,107:194]
```

```
img[0:74,0:87] = watch_face
```

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

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

Esto funciona en este caso para la imagen seleccionada, pero depende del tamaño de la imagen que se utilice.

## Resultados:

```
*IDLE Shell 3.13.2*
File Edit Shell Debug Options Window Help


[255 255 255]
[255 255 255]

[[255 255 255]
[255 255 255]
[255 255 255]
...
[255 255 255]
[255 255 255]
[255 255 255]]

...

[[255 255 255]
[255 255 255]
[255 255 255]
...
[255 255 255]
[255 255 255]
[255 255 255]]

[[255 255 255]
[255 255 255]
[255 255 255]
...
[255 255 255]
[255 255 255]
[255 255 255]]]
(168, 300, 3)
151200
uint8
```



## Objetivo:

Realizar operaciones aritméticas simples con imágenes al igual que explicar el funcionamiento de estas.

## Código:

```
import cv2
import numpy as np

# 500 x 250
img1 = cv2.imread('3D-Matplotlib.png')
img2 = cv2.imread('mainsvmimage.png')

#add = img1+img2
add = cv2.add(img1,img2)

weighted = cv2.addWeighted(img1, 0.6, img2, 0.4, 0)

rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols ]

img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

ret, mask = cv2.threshold(img2gray, 220, 255, cv2.THRESH_BINARY_INV)

mask_inv = cv2.bitwise_not(mask)

img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)

img2_fg = cv2.bitwise_and(img2,img2,mask = mask)

dst = cv2.add(img1_bg,img2_fg)
img1[0:rows, 0:cols ] = dst

cv2.imshow('add',add)
cv2.imshow('weighted',weighted)
cv2.imshow('res',img2)
cv2.imshow('mask_inv ',mask_inv)
cv2.imshow('img1_bg',img1_bg)
cv2.imshow('img2_fg',img2_fg)
cv2.imshow('dst',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Comentarios:

Necesitaremos dos imágenes del mismo tamaño para comenzar, luego más adelante una imagen más pequeña y una más grande.

Para realizar nuestra Suma (+/ADD) podemos realizarlo con dos comandos posibles los cuales al ejecutar el programa veremos que se realiza la misma operacion pero con resultados diferentes:

```
add = img1+img2
```

```
add = cv2.add(img1,img2)
```

Para el método addWeighted, los parámetros son la primera imagen, el peso, la segunda imagen, ese peso y, finalmente, gamma, que es una medida de la luz. Lo dejaremos en cero por ahora.

```
weighted = cv2.addWeighted(img1, 0.6, img2, 0.4, 0)
```

Quiero poner el logotipo en la esquina superior izquierda, así que creo un ROI

```
rows,cols,channels = img2.shape
```

```
roi = img1[0:rows, 0:cols ]
```

Ahora crea una máscara de logotipo y crea su máscara inversa

```
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
```

Agregar un threshold

```
ret, mask = cv2.threshold(img2gray, 220, 255, cv2.THRESH_BINARY_INV)
```

```
mask_inv = cv2.bitwise_not(mask)
```

Ahora oscurezca el área del logotipo en ROI

```
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)
```

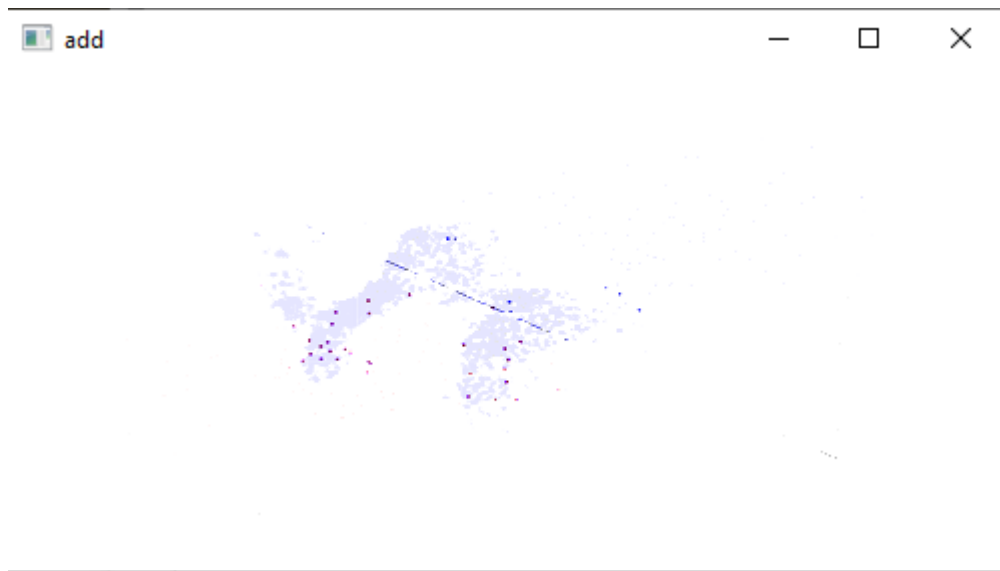
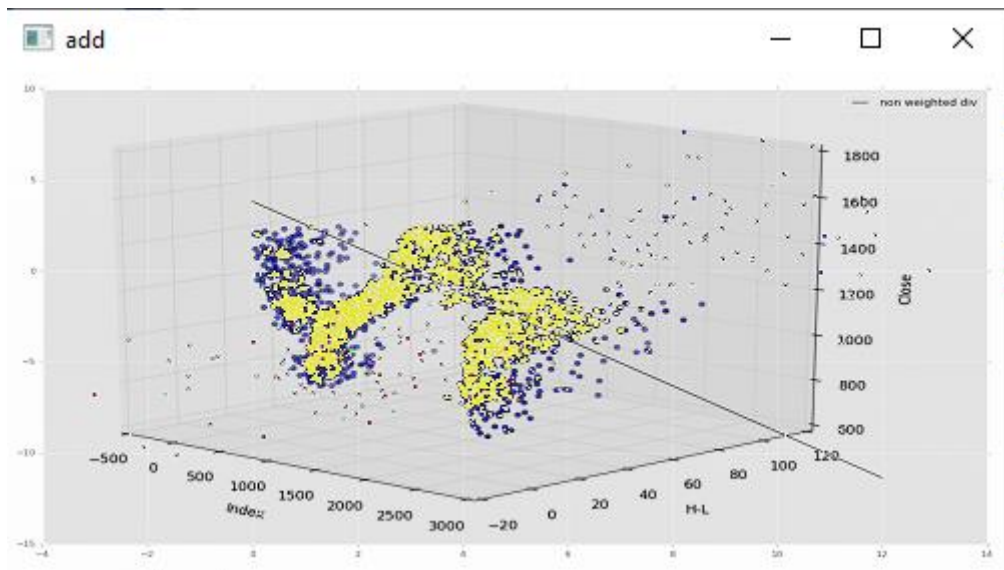
Tome solo una región del logotipo de la imagen del logotipo.

```
img2_fg = cv2.bitwise_and(img2,img2,mask = mask)
```

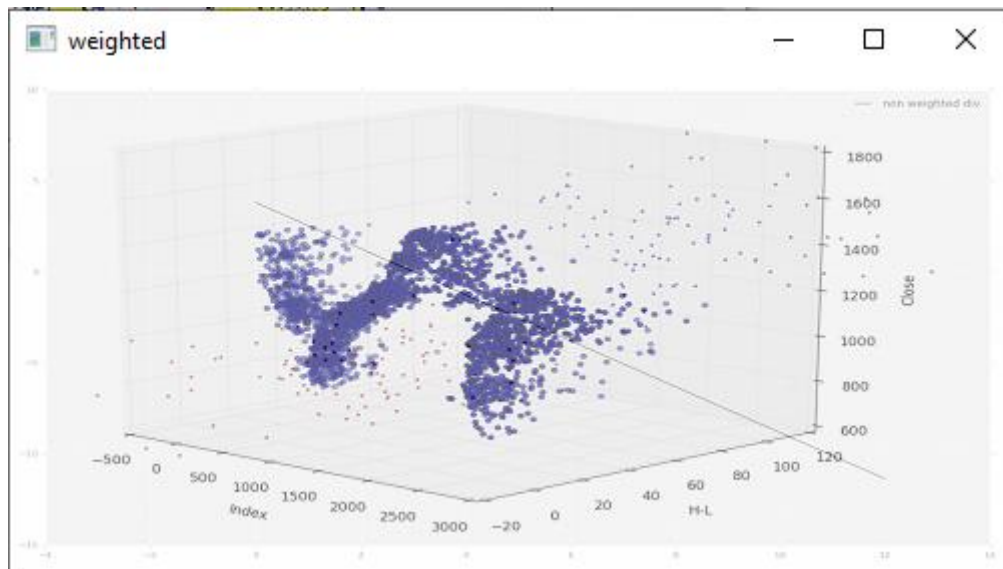


Resultados:

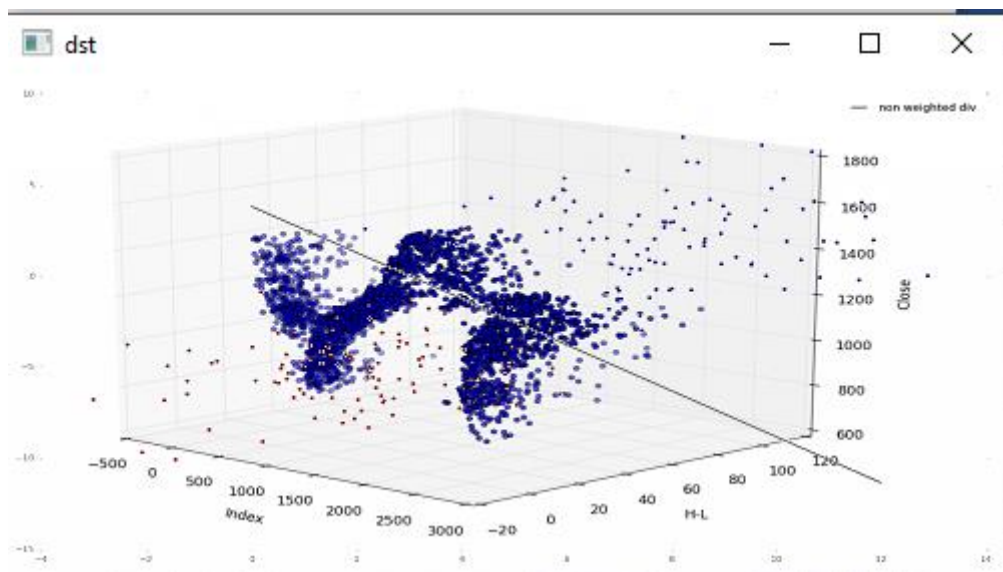
ADD



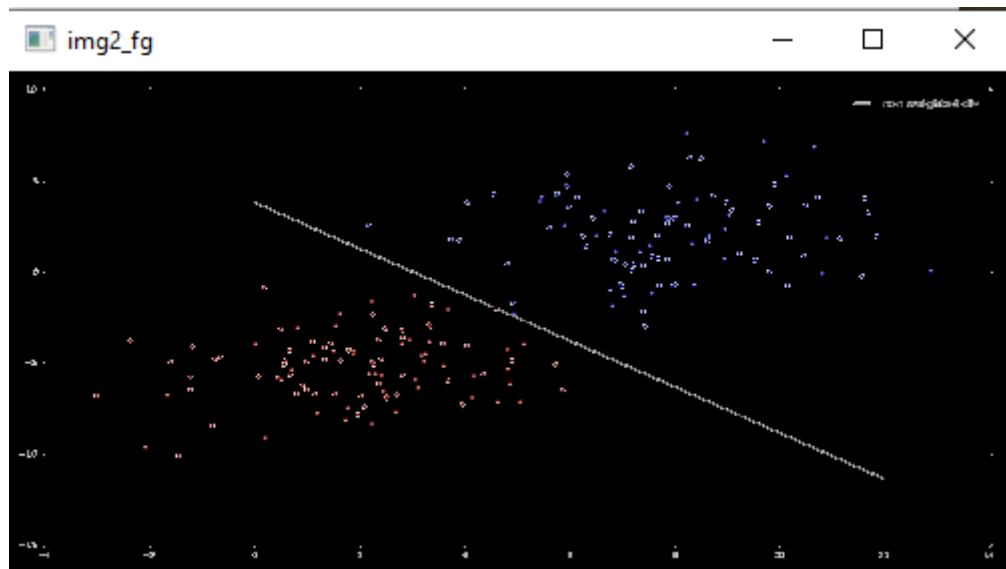
WEIGHTED



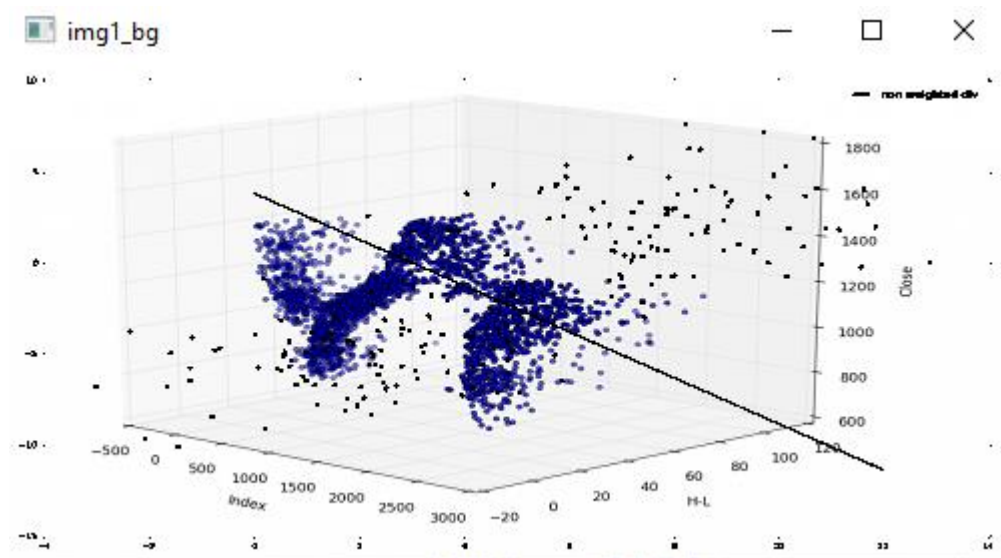
DST



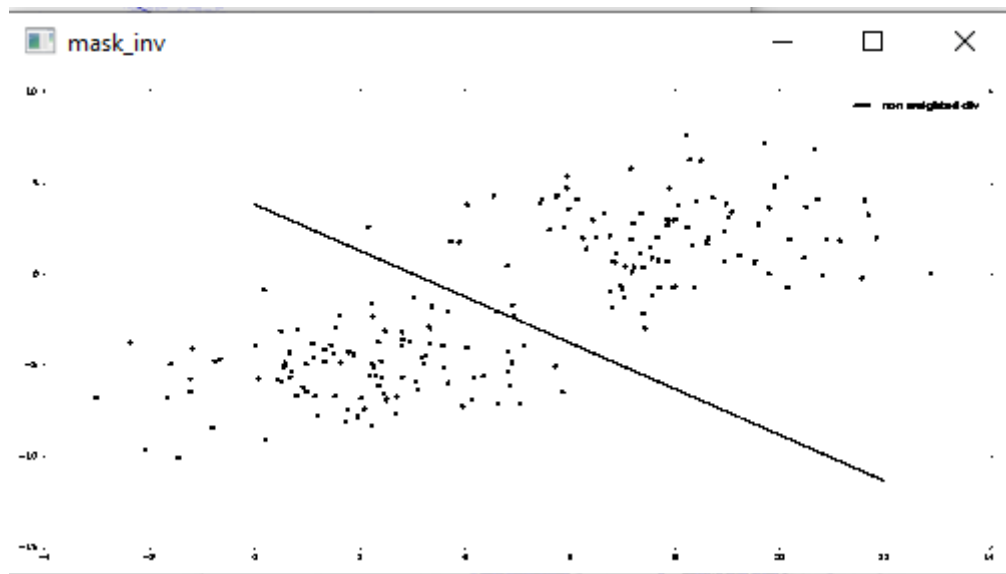
Img2\_fg



Img1\_bg



Mask\_inv



RES

