DETECCIÓN DE PUNTOS CLAVES Y DESCRIPTORES CON ORB

El método ORB (Oriented FAST and rotated BRIEF) es una unión del detector de puntos claves FAST y el descriptor BRIEF, pero con mejorar para su rendimiento.

Para encontrar los puntos clave ORB hace uso del método FAST, luego aplica la detección de esquinas de Harris para encontrar los *N* puntos principales entre ellos. También utiliza la pirámide multiescala de imágenes para producir características, estas pirámides se basan en tomar una imagen como principal y mediante esa imagen se redimensiona a menores escalas formando así una pirámide en la cual se toman todas sus características desde la imagen base hasta la última.

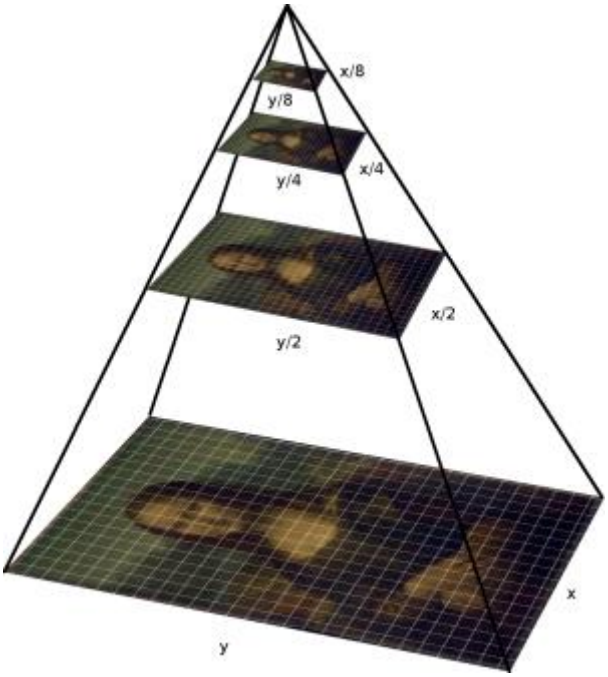


Figura 1: Pirámide multiescala. [1]

El problema del método FAST es que no calcula la orientación, para esto los creadores de ORB hicieron una modificación la cual consiste en calcular la intensidad ponderada del centroide del parche con la esquina ubicada en el centro, la orientación es dada a partir de la dirección del vector que se ubica desde la esquina del centro hasta el centroide. Para la invariación de rotación, los momentos se calculan a partir de las coordenadas *x* e *y*, las cuales deben de estar en una región circular de radio *r*, en donde *r* es el tamaño del parche.

Ahora para los descriptores lo que ORB hace es dirigir los descriptores de BRIEF de acuerdo a la orientación de los puntos claves. Para cualquier conjunto de *n* pruebas binarias localizadas en (x_i, y_i) se define una matriz dada por *2n*, y *S* es la que contiene las coordenadas de los pixeles. Ahora haciendo uso de la orientación del parche, θ, se encuentra la matriz de rotación para rotar *S* y así obtener la versión dirigida o rotada de S_θ.

Por último, para calcular los descriptores ORB discretiza el ángulo en incremento de $2\pi/30$ (12 grados) y crea una tabla de búsqueda de patrones BRIEF pre calculados y mientras la orientación del punto clave, θ, sea consistente en todas las vistas el conjunto correcto de S_θ será usado para calcular su descriptor.

Tal vez esto sea un poco raro al momento de verlo por primera vez, pero cuando lo pongamos en práctica va a tomar más lógica toda la teoría explicada.

Para este proyecto usaremos la librería de NumPy y Cv2, empezamos importando las librerías en nuestro código e inicializando nuestra captura de video con la función .videoCapture(), también leemos la imagen que usaremos con la función .imread() y como parámetro el nombre de la imagen guardada en la carpeta del código.

```
import numpy as np
import cv2

Cap= cv2.VideoCapture(0)
Img = cv2.imread("BOOK_IMAGE.jpg")
```

Crearemos nuestro ciclo while el cual contendrá la mayoría del código, después de esto convertiremos tanto nuestros frames previamente leídos con la función .read() como nuestra imagen a escala de grises, para esto usaremos la función .cvtColor(), como parámetros para esta función le daremos la variable del elemento a usar seguido del modo de cambio de escala, por defecto Opencv usa la escala BGR.

```
while True:
    Rec, Frame = Cap.read()

    #GRAY SCALE TO FRAMES AND IMAGE
    Gray_Frames=cv2.cvtColor(Frame, cv2.COLOR_BGR2GRAY)
    Gray_Img=cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)
```

Ahora para inicializar nuestro detector ORB debemos crear una variable la cual contenga una cantidad de puntos a detectar que deseamos , y con la función .ORB_create(), a la cual le daremos como parámetro la variable con la cantidad de puntos, tendremos ya creada nuestra inicialización.

```
#NUMBERS OF POINTS TO DETECT
Number=500
#START THE FUNCTION .ORB_create(NUMBERS OF POINTS)
Orb = cv2.ORB_create(Number)
```

Siguiendo con el código pasaremos a poner en práctica todo la teoría que vimos anteriormente, pero de una forma sumamente sencilla, en solo dos líneas de código podemos hacer esto gracias a el método ORB que hace todo el cálculo matemático de forma automática.

Para esto usaremos la función .detectAndCompute() a la cual le daremos como parámetro el nombre de la variable creada para la conversión a escala de grises.

```
#DETECT THE POINTS WITH FUNCTION .detectAndComputer(Img, None )
CAUTION: IN THE VARIABLES OF THE BEGINNING ALWAYS PUT KEYPOINT AND
DESCRIPTOR AS VARIABLES
Keypoints1, descriptor1= Orb.detectAndCompute(Gray_Frames,None)
Keypoints2, descriptor2= Orb.detectAndCompute(Gray_Img,None)
```

En esta parte del código hay que dejar claro que se hace exactamente, primero si vemos tenemos varias variables las cuales son: Keypoints1, descriptor1 y Keypoints2, descriptor2, si recordamos un poco, habíamos explicado que el método ORB calcula tanto los puntos clave como los descriptores por eso estas dos variables, debes tener cuidado ya que si no incluyes estas dos variables tu programa puede presentar errores, es recomendable siempre incluir ambas variables. Ahora si podemos observar en la función .detectAndCompute() tenemos la variable Orb esto es porque la variable contiene la inicialización del método con la cantidad de puntos a detectar y la función hace lo que dice su nombre, detecta y calcula esos 500 puntos que le dimos anteriormente, aquí es donde se pone en práctica lo visto, a partir de esos 500 puntos se calcula todo lo visto anteriormente, lo hace más sencillo, sin tener que usar formulas escritas en el código, pero era importante que supieras cómo funcionaba teóricamente para así comprender bien el porqué de la función, el poner dos variables o la cantidad de puntos a detectar.

Una vez teniendo en claro la forma en la que funciona y el porqué del código anterior, podemos obtener los valores de los descriptores imprimiéndolos en pantalla.

```
print("DESCRIPTOR1",descriptor1)
print("DESCRIPTOR2",descriptor2)
```

Ya tenemos los descriptores, ahora dibujaremos los puntos clave mediante la función .drawKeypoints() la cual le pasaremos como parámetros lo siguiente:

- Imagen en la cual dibujaremos los puntos clave.
- La variable donde se almacenan los puntos clave.
- La imagen de salida.
- Color de los puntos clave.
- Banderas que configuran características de dibujo.

Para dibujar los puntos claves se pueden hacer de dos formas, una es usando la librería NumPy en la parte de la imagen de salida y la otra es solo dándole un valor de 0.

```
#DRAW THE POINT DETECTED
Frames_Display= cv2.drawKeypoints(Frame, Keypoints1, 0, color=(255,0,0),flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)
Img_Display= cv2.drawKeypoints(Img, Keypoints2, 0, color=(255,0,0),flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)
```

```
#DRAW THE POINT DETECTED
Frames_Display= cv2.drawKeypoints(Frame, Keypoints1, np.array([]), color=(255,0,0),flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)
Img_Display= cv2.drawKeypoints(Img, Keypoints2, np.array([]), color=(255,0,0),flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT)
```

Una vez teniendo esto solo queda mostrarlos en pantalla con la función .imshow() y como parámetros el nombre que tendrá nuestra ventana y la variable a mostrar con los puntos dibujados.

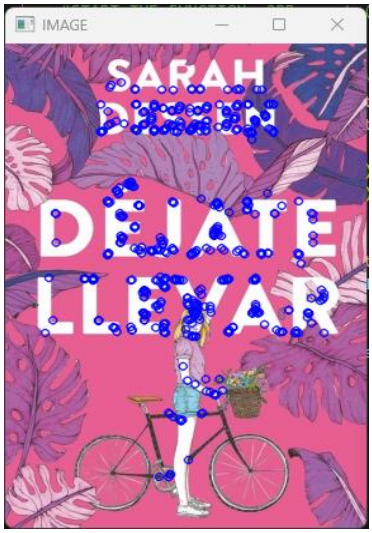
```
#SHOW THE POINTS IN SCREEN
cv2.imshow("VIDEO CAPTURE",Frames_Display)
cv2.imshow("IMAGE",Img_Display)
```

Ya para terminar solo tenemos que crear nuestro valor de salida con la función .waitKey(), eliminar nuestra captura de video y ventanas.

```
    t = cv2.waitKey(1)
    if t == 27:
        break


Cap.release()
cv2.destroyAllWindows()
```

Como resultado de este código se obtendrá una imagen con diversos círculos que son los puntos claves y la impresión de los valores de los descriptores como se muestra en las siguientes imágenes, en la parte derecha esta la imagen tomada con la cámara y en la parte izquierda la imagen guardada y por ultimo los valores de los descriptores.



```
DESCRIPTOR1 [[ 98  43  96 ... 204  94 206]
 [154  28 110 ...  13  61 223]
 [ 10 159  34 ...  76 181 251]
 ...
 [ 70 150 181 ... 135  37 112]
 [ 17   9 238 ... 232  57 112]
 [ 67 252 133 ...  25  25  30]]
DESCRIPTOR2 [[236 212 252 ... 121  92  11]
 [222 107 158 ... 119  47  75]
 [ 34 165  64 ...  10 208 128]
 ...
 [ 25 246 217 ... 130  33  41]
 [ 72 178  91 ...   2  35 177]
 [  1 225  27 ...   2  35 169]]
DESCRIPTOR1 [[ 32 205  58 ...  27 116 170]
 [131 221 118 ... 253 241 223]
 [158  30 110 ...  77  61 215]
 ...
 [ 34 166 226 ...  74 112 143]
 [229  81 189 ...  57 214  59]
 [102 242 145 ...  41  23  87]]
DESCRIPTOR2 [[236 212 252 ... 121  92  11]
 [222 107 158 ... 119  47  75]
 [ 34 165  64 ...  10 208 128]
 ...
...
```

Si llegaste hasta esta parte te agradezco el tiempo que me regalaste y espero te haya ayudado con tus dudas o inquietudes de aprender más, esto solo es un pequeño fragmento de explicación si quieres seguir aprendiendo más puedes visitar las siguientes páginas que te ayudaran a seguir aprendiendo y mejorando en el tema de la visión artificial y Python.

1.- https://omes-va.com/

2.- https://www.pythonpool.com/

3.- https://hetpro-store.com/TUTORIALES/

4.- https://learnopencv.com/

Lo principal es la sabiduría; adquiere sabiduría, Y con todo lo que obtengas adquiere inteligencia. Proverbios 4:7

REFERENCIAS

Tutor de programación. (2017). Pirámides de Imágenes con OpenCV. [Figura 1]. Recuperado de http://acodigo.blogspot.com/2017/02/piramides-de-imagenes-con-opencv.html