DETECTION OF KEYPOINTS AND DESCRIPTORS WITH ORB

The ORB (Oriented FAST and rotative BRIEF) method is a combination of the FAST key points detector and the BRIEF descriptor, but with improved efficiency.

To find the key points, ORB makes use of the FAST method then it applies the Harris corners detection to find the *N* main points among them. It also uses the multiscale pyramids of images to make features, these pyramids are based on take an image as main and through this image is resized to a smaller scale, forming like this a pyramid in which are taking all its features from the base image to the last one.

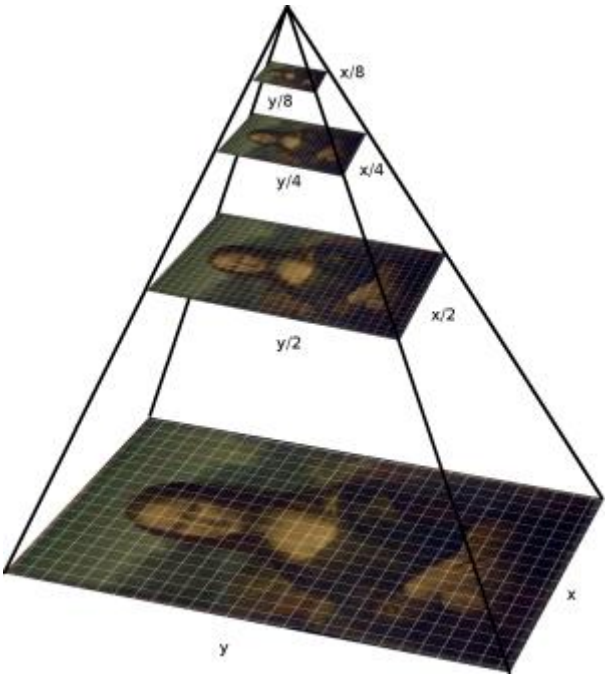


Figure 1: Multiscale pyramid.

The problem of the FAST method is that it does not calculate the orientation to this the creators of ORB made a modification which consists of calculating the weighted intensity of the centroid of the patch with the corner located in the center, the orientation is given by vector address that is located from the corner of the center to the centroid. For the invariance of rotation, the moments are calculated from the coordinates *x* and *y*, which should of be in a circular region of radius *r*, where *r* is the patch size.

Now to the descriptors what ORB does is to lead the descriptors of BRIEF according to the orientation of the key points. For any set of *n* binary tests located in (x_i, y_i) it will define a matrix given by *2n*, and *S* contains the coordinates of the pixels. Now doing use the orientation of the patch, θ, matrix rotation is found to rotate *S* and so get the version lead or rotated of S_θ.

Finally, to calculate the descriptors ORB discretize the angle in increments of $2\pi/30$ (12 degrees) and create a lookup table of BRIEF patterns precalculated and while the orientation of the key point, θ, will be constant in all the views of conjunct correct of S_θ it will be used to calculate its descriptor.

Maybe it will be a little rare at moment of seeing it the first time, but when put in practice it will take more logic all the theory explained.

For this project we will use the libraries of NumPy and Cv2, we start importing the libraries in our code and initializing our video capture with the function .videoCapture(), we also read the image that we will use with the function .imread() and as parameter the name of the image saved in the folder of the code.

```
import numpy as np
import cv2

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

We create our while-loop that contains most of the code, then of this we will convert both our frames read with the function .read() and our image to grayscale, to do this we will use the function .cvtColor(), as parameters to this function we will give the variable of the element to use followed by the mode of scale change, by default OpenCV uses the BGR scale.

```
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)
```

Now to initialize our detector ORB we have to create a variable that will contain a number of points to detect that we want, and with the function .ORB_create(), which we will give as parameters the variable with the number of points, we will have already created our initialization.

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

Following with the code we will pass to put in practice all the theory that we have seen before but in a very simple way, in two lines of code we can do this thanks to the ORB method that does all the mathematical compute of automatic way.

For this we will use the function .detectAndCompute() which we will give as parameter the name of the variable created for the conversion to grayscale.

```
#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)
```

In this part of the code, it is necessary to make clear that it does exactly, first if we see to have some variables that are: Keypoints1, descriptor1 and Keypoints2, descriptor2, if we remember a little, we had explained that the method ORB calculates both the key points and the descriptors for these two variables, you must be careful because if you do not add these two variables you program can present errors, it is recommended to always include both variables. Now if we can observe in the function .detectAndCompute() we have the variable Orb this is because the variable contains the initialization of the method with the number of points to detect and the function does that say its name, detect and compute these 500 points that we gave it before, here is where what has been seen is put into practice, from these 500 points is calculated all the things seen before, it makes easier without having to use formulas writes in the code, but it was important that you knew how worked theoretically to like this understand well the reason of the function, why to put two variables or the number of points to detect.

Once it is clear the form in which it works and why of the code before, we can obtain the values of the descriptors print it in the screen.

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

We already have the descriptors, now we draw the key points with the function .drawKeyPoints() which we give as parameters the next:

- Image on which we will draw the key points.
- Variable where the key points will be stored.
- Output image.
- Color of the key points.
- Flags that will configure the drawing properties.

To draw the key points, it can be done of two ways, one way is using the NumPy library in the output image and the other is just giving it a value of 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)
```

Once we have this, we have to show in screen with the function .imshow() and as parameters the name that our window will have and the variable to show with the points drawn.

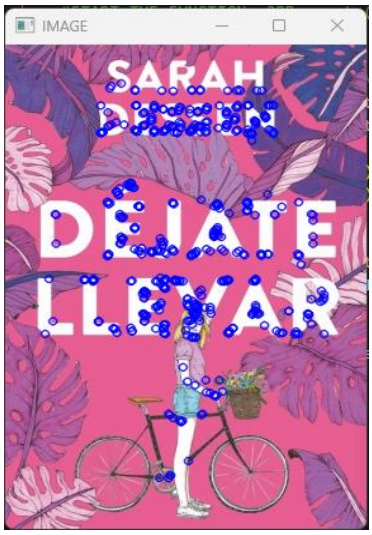
```
#SHOW THE POINTS IN SCREEN
cv2.imshow("VIDEO CAPTURE",Frames_Display)
cv2.imshow("IMAGE",Img_Display)
```

To finish we only have to create our exit value with the function .waitKey(), delete our video captures and the windows.

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


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

As result of this code we will obtain an image with different circles that are the key points and the value prints of the descriptors as it is shown in the next images, on the right is the image token with the camera and on the left the saved image, and finally the descriptor values.



```
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]
 ...
...
```

If you got to this part, I thank you for the time that you gave me and I hope that I have helped to you with you doubts and concerns of learning more, this is just a small fragment of explain if you want to continue learning and improve you can visit these pages that will help you in the topic of artificial vision and Python.

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

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

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

4.- https://learnopencv.com/

Wisdom is the principal thing; therefore get wisdom: yea, with all thou hast gotten get understanding. Proverbs 4:7

REFERENCES

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