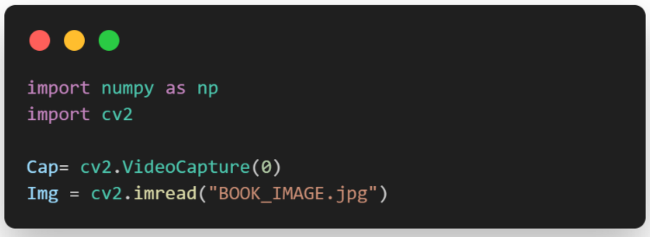
# MATCHING SIMILAR FEATURES USING THE ORB METHOD

Before of start, if you want to know how the ORB method works and understand better the first lines of code I recommended to you that you will visit the next repository https://github.com/Enock13/DETECTION-OF-KEYPOINTS-AND-DESCRIPTORS-WITH-ORB.git.

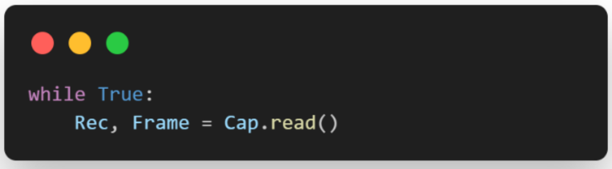
In this repository you will find the explanation step by step of the function of the ORB method, as well as the identification of the key points and descriptors.

Once understood the before we can pass to the part of matching the similar features, to this code we will use the libraries NumPy and OpenCV, once imported these libraries we will initialize our video capture with the function .videoCapture() and we will read our reference image too, previously saved in the code folder, with the function .imread().



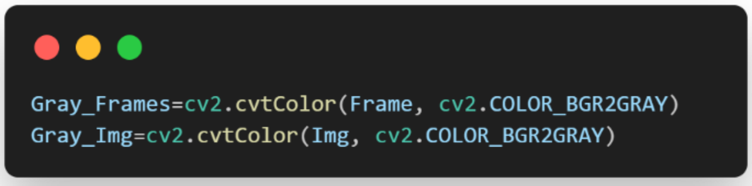
```python
import numpy as np
import cv2

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

Now we will create our while-loop it will contain the most important part of the code to next read our frames obtained from the video capture through the .read() function.



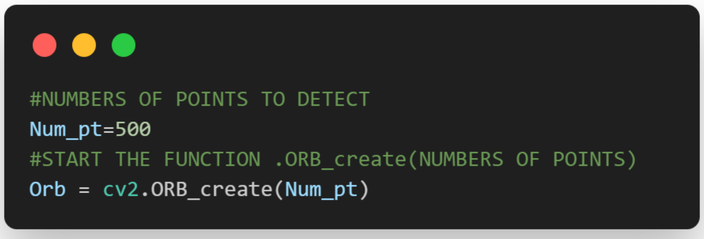
```python
while True:
    Rec, Frame = Cap.read()
```

Once read the frames we will pass to convert to grayscale, both the frames and the reference image, this through the .cvtColor() function which we will give as parameters the name of the variables to convert, as well as the conversion mood which will be from BGR (color scale that OpenCV uses by default) to GRAY.



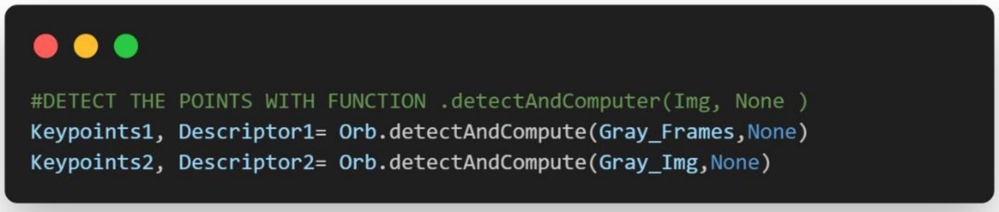
```python
Gray_Frames=cv2.cvtColor(Frame, cv2.COLOR_BGR2GRAY)
Gray_Img=cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)
```

Before of start the ORB method, we will give the points numbers that we want and once having this we can initialize the method with the function .ORB_create() and as parameters the variable where the numbers of points are store.



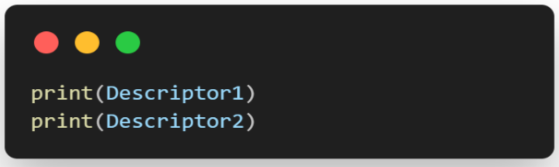
```
#NUMBERS OF POINTS TO DETECT
Num_pt=500
#START THE FUNCTION .ORB_create(NUMBERS OF POINTS)
Orb = cv2.ORB_create(Num_pt)
```

Now we will pass to save our key points and descriptors with the function .detetctAndCompute() and as parameter the variable designated to the grayscale that we previously converted, this will be done both for the frames and the reference image.



```
#DETECT THE POINTS WITH FUNCTION .detectAndComputer(Img, None )
Keypoints1, Descriptor1= Orb.detectAndCompute(Gray_Frames,None)
Keypoints2, Descriptor2= Orb.detectAndCompute(Gray_Img,None)
```

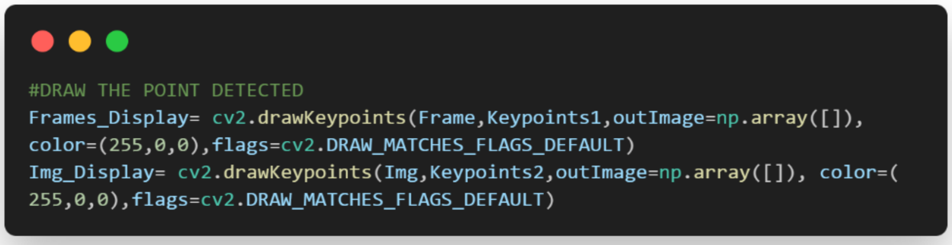
If we want to know the values for the descriptors, we just have to print them, and this way we get the values.



```
print(Descriptor1)
print(Descriptor2)
```

For show the key points, we have to draw first these key points, for that we will use the function .drawKeypoints() and as parameters the next elements:

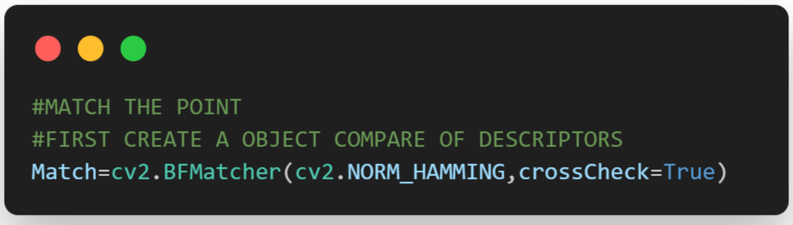
- Image on which the points are displayed.
- Detected key points.
- Output image.
- Color of the points.
- Flags that configure the drawing functions.



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

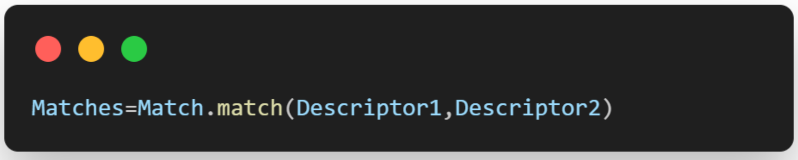
To be able to match the features is necessary to create an object with the function .BFMatcher(), this function needs of two parameters, which are:

- normType, in this we specify the distance measurement to use, as we used the ORB method, we will use the measurement .NORM_HAMMING.
- croosCheck, this boolean parameter evaluates, if it is True, the points that match in both images.



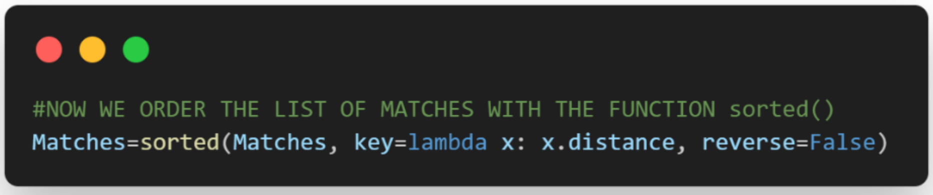
```
#MATCH THE POINT
#FIRST CREATE A OBJECT COMPARE OF DESCRIPTORS
Match=cv2.BFMatcher(cv2.NORM_HAMMING,crossCheck=True)
```

Once we have created the object created, we are going to use the variable where we have stored the object with its parameters to use them in the descriptors of the reference image and the video capture, this through the function .match() and as parameters the descriptors previously mentioned, since these have the coordinates of each characteristic point that will be to use to do the matching.



```
Matches=Match.match(Descriptor1,Descriptor2)
```

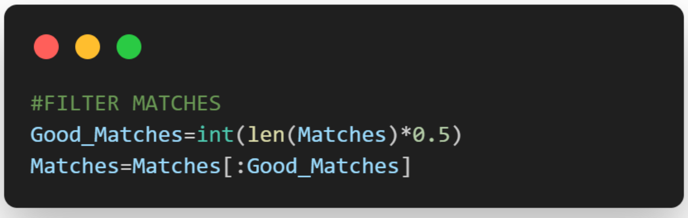
Now, that match that we have created we are going to sort, this will be done with the function sorted() of Python, this function will sort the matches according to the sorted that we will give it as parameter. First we give it the variable we want to be sorted in this case will be Matches because this variable is the one that contain the descriptors matching, we will see that the parameter key has an equal, this equal is given for "lambda x: x.distance" that it will make is take the list given for the variable Matches and sorted in function of the distance, to minor distance better matching, and finally the parameter boolean revers it will be False, if we put it in True the list will be sorted from major to minor distance.



```python
#NOW WE ORDER THE LIST OF MATCHES WITH THE FUNCTION sorted()
Matches=sorted(Matches, key=lambda x: x.distance, reverse=False)
```

After having the sorted, we are going to filter those matching that we have until now, this will be done by the integer value that will be obtained of the 50% of the total matches.

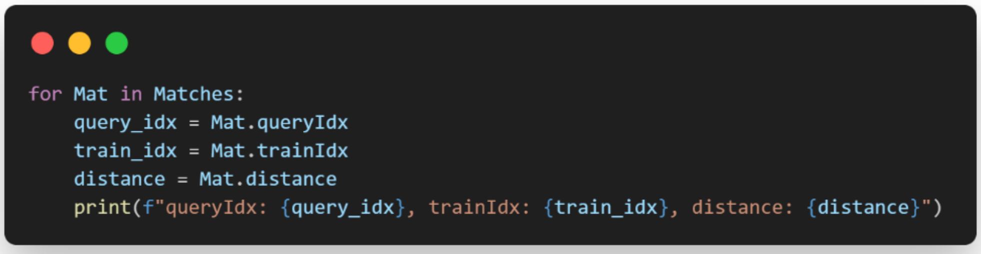
Then this integer value we will store of new in the variable Matches, giving it as value the list of this filtration that we have just finished. Finally, the variable Matches will have the middle of the matchings made this with the proposal of obtaining the better matchings and deleting the matchings without recommended distances.



```python
#FILTER MATCHES
Good_Matches=int(len(Matches)*0.5)
Matches=Matches[:Good_Matches]
```

The next part is optional, but if we want to know the values of the matches, we could create a for-loop in which we get three values which are:

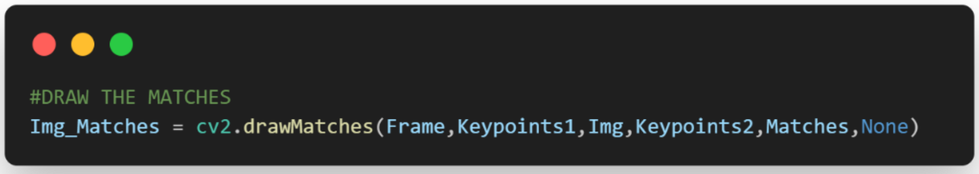
- query_idx: it is the index to have the descriptors in the set of characteristics of the query image.
- train_idx: it is the index to have the descriptors in the set of characteristics of the train image.
- distance: it is the distance between the matching descriptors.



```python
for Mat in Matches:
    query_idx = Mat.queryIdx
    train_idx = Mat.trainIdx
    distance = Mat.distance
    print(f"queryIdx: {query_idx}, trainIdx: {train_idx}, distance: {distance}")
```

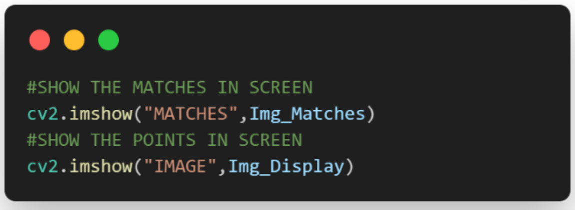
Already having done all this made and continuing with the code, we will have to draw these matches found, this through the function .drawMatches() that we will give as parameters the next:

- Frames variable.
- Frames key points variable.
- Reference image variable.
- Key points variable of the reference image.
- Variable containing the matches.
- Output image (in this case None).



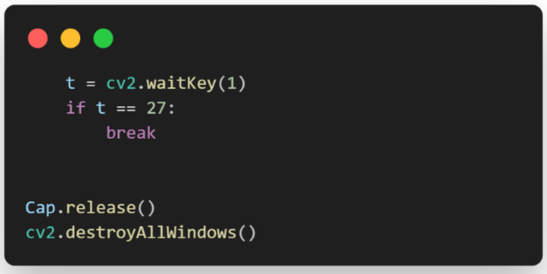
```
#DRAW THE MATCHES
Img_Matches = cv2.drawMatches(Frame,Keypoints1,Img,Keypoints2,Matches,None)
```

With the matches drawn, we can show them with the function .imshow() and as parameters the name that the window will have and the variable of that we want to show.



```
#SHOW THE MATCHES IN SCREEN
cv2.imshow("MATCHES",Img_Matches)
#SHOW THE POINTS IN SCREEN
cv2.imshow("IMAGE",Img_Display)
```

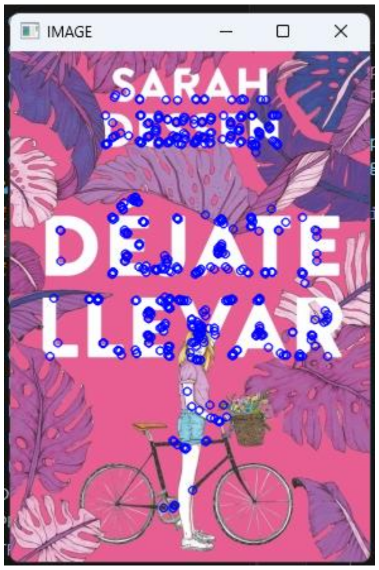
To end our code, we will create the exit command with the function .waitKey() and the key "ESC" of our keyboard, which corresponds to the number 27 in the ASCII system, also we will delete our video capture with the function .release() and our windows with .destroyAllWindows() to end the program.

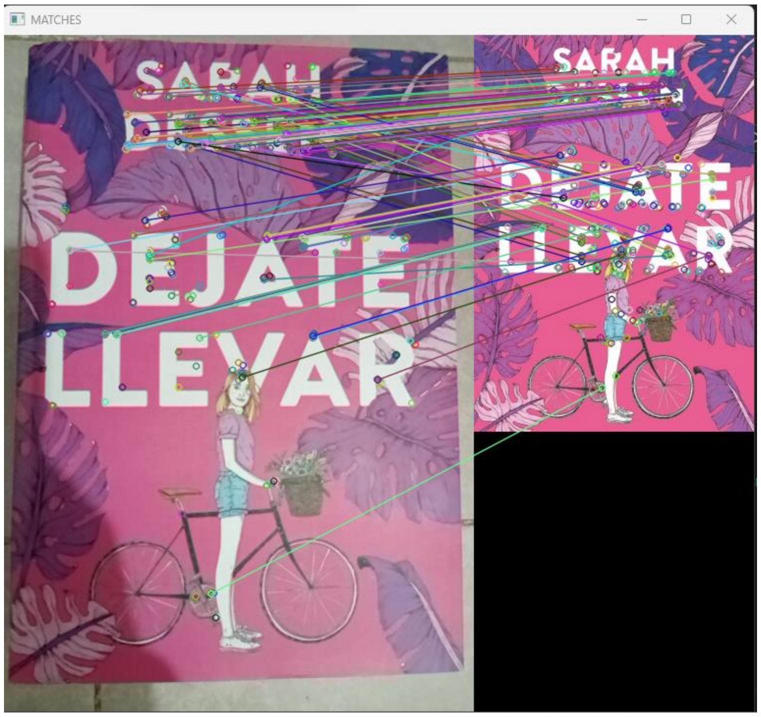


```
    t = cv2.waitKey(1)
    if t == 27:
        break


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

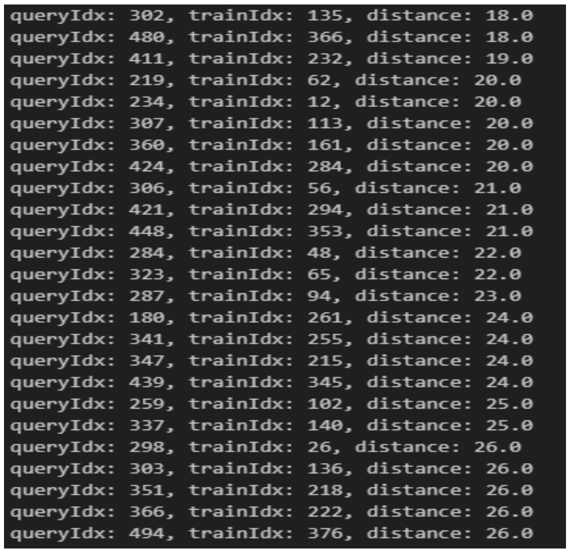
## RESULTS

Now, it will be shown the results obtained in this code, first with 500 points and then the same test, but with 200 points to detect, also it will be shown the values of the end match.



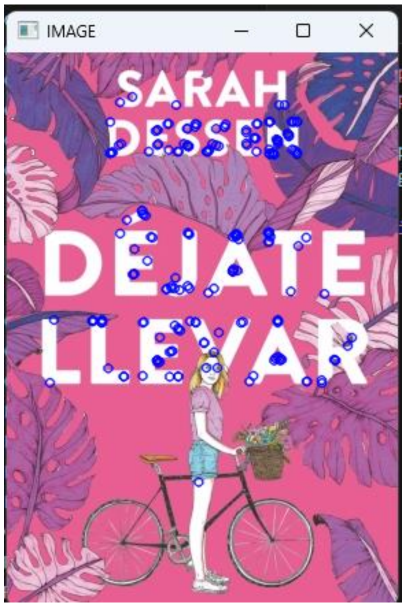


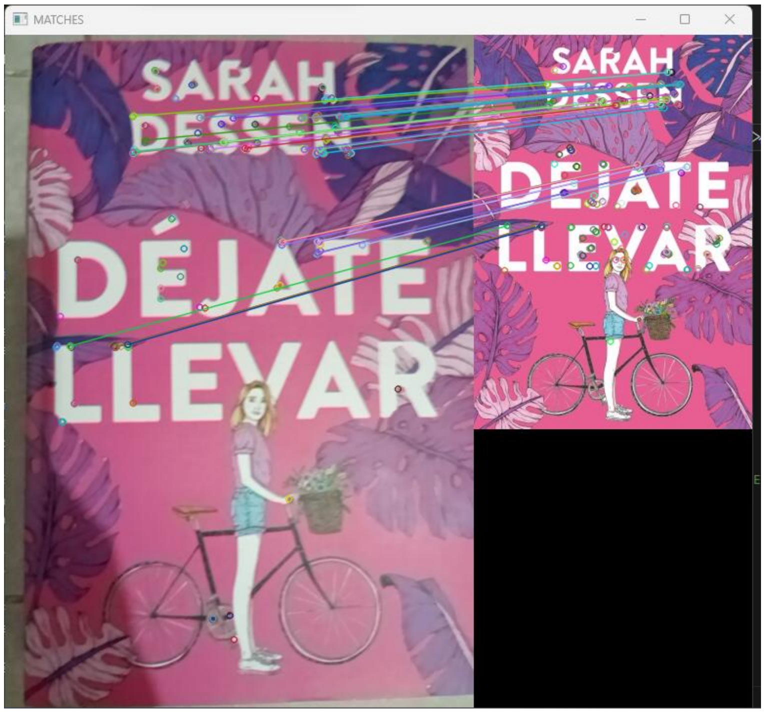
From the right side we can see the key points of the image and the right side the matching, now we see the results of the values of this matching, we see that in the part of the distance they are from minor to major, this indicate that the function sorted work correctly.

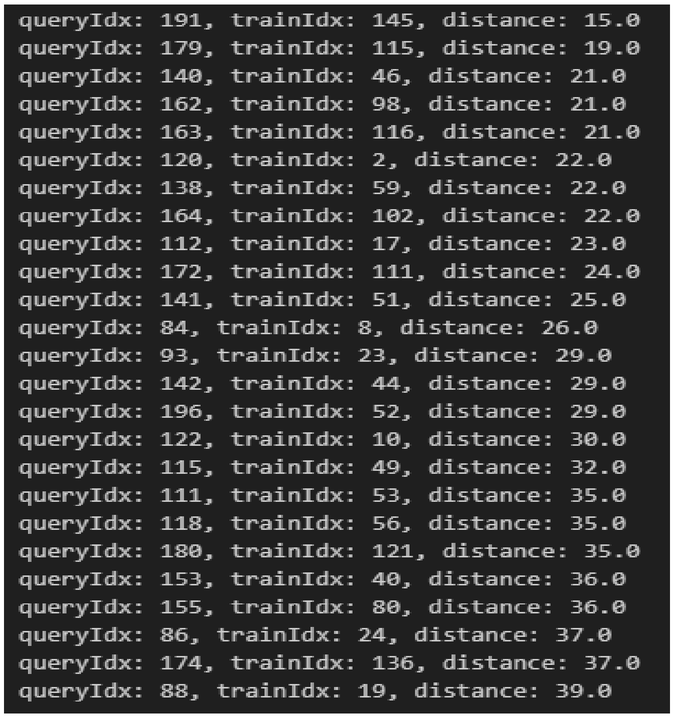


```
queryIdx: 302, trainIdx: 135, distance: 18.0
queryIdx: 480, trainIdx: 366, distance: 18.0
queryIdx: 411, trainIdx: 232, distance: 19.0
queryIdx: 219, trainIdx: 62, distance: 20.0
queryIdx: 234, trainIdx: 12, distance: 20.0
queryIdx: 307, trainIdx: 113, distance: 20.0
queryIdx: 360, trainIdx: 161, distance: 20.0
queryIdx: 424, trainIdx: 284, distance: 20.0
queryIdx: 306, trainIdx: 56, distance: 21.0
queryIdx: 421, trainIdx: 294, distance: 21.0
queryIdx: 448, trainIdx: 353, distance: 21.0
queryIdx: 284, trainIdx: 48, distance: 22.0
queryIdx: 323, trainIdx: 65, distance: 22.0
queryIdx: 287, trainIdx: 94, distance: 23.0
queryIdx: 180, trainIdx: 261, distance: 24.0
queryIdx: 341, trainIdx: 255, distance: 24.0
queryIdx: 347, trainIdx: 215, distance: 24.0
queryIdx: 439, trainIdx: 345, distance: 24.0
queryIdx: 259, trainIdx: 102, distance: 25.0
queryIdx: 337, trainIdx: 140, distance: 25.0
queryIdx: 298, trainIdx: 26, distance: 26.0
queryIdx: 303, trainIdx: 136, distance: 26.0
queryIdx: 351, trainIdx: 218, distance: 26.0
queryIdx: 366, trainIdx: 222, distance: 26.0
queryIdx: 494, trainIdx: 376, distance: 26.0
```

Now the same test, but with 200 points and with this we have checked the efficiency of the code.







```
queryIdx: 191, trainIdx: 145, distance: 15.0
queryIdx: 179, trainIdx: 115, distance: 19.0
queryIdx: 140, trainIdx: 46, distance: 21.0
queryIdx: 162, trainIdx: 98, distance: 21.0
queryIdx: 163, trainIdx: 116, distance: 21.0
queryIdx: 120, trainIdx: 2, distance: 22.0
queryIdx: 138, trainIdx: 59, distance: 22.0
queryIdx: 164, trainIdx: 102, distance: 22.0
queryIdx: 112, trainIdx: 17, distance: 23.0
queryIdx: 172, trainIdx: 111, distance: 24.0
queryIdx: 141, trainIdx: 51, distance: 25.0
queryIdx: 84, trainIdx: 8, distance: 26.0
queryIdx: 93, trainIdx: 23, distance: 29.0
queryIdx: 142, trainIdx: 44, distance: 29.0
queryIdx: 196, trainIdx: 52, distance: 29.0
queryIdx: 122, trainIdx: 10, distance: 30.0
queryIdx: 115, trainIdx: 49, distance: 32.0
queryIdx: 111, trainIdx: 53, distance: 35.0
queryIdx: 118, trainIdx: 56, distance: 35.0
queryIdx: 180, trainIdx: 121, distance: 35.0
queryIdx: 153, trainIdx: 40, distance: 36.0
queryIdx: 155, trainIdx: 80, distance: 36.0
queryIdx: 86, trainIdx: 24, distance: 37.0
queryIdx: 174, trainIdx: 136, distance: 37.0
queryIdx: 88, trainIdx: 19, distance: 39.0
```

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**