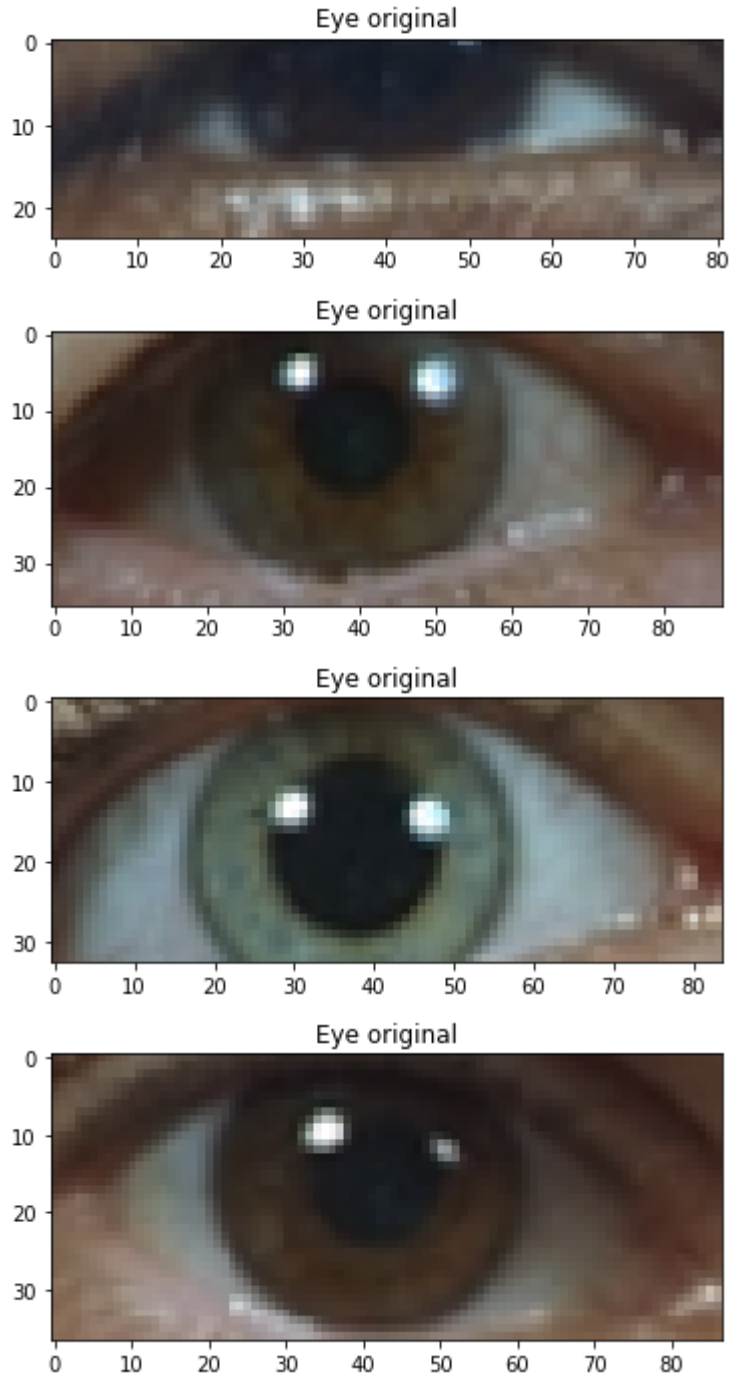
Opencv: How can I get the eye color

Asked 3 years, 11 months ago Active 3 months ago Viewed 2k times

0

I am using dblib to get the eyes of a face. Below are some examples of the results.



I have tried several methods to accomplish the objective. For instance, I tried to detect the center of the eye based on this project; from that, it would be easy to detect the pupil and the iris, however, I did not achieve good results. I also have tried to use Hough Circles but in some cases the results are quite bad.

My best bet is to detect the pupil, which is the only part of the eye with a common color (black) for every eye. I would like to get some ideas to do so.

My first idea is to set a region (between 20 and 60 in the x axis), then, in gray-scale, make the dark pixels (less than 25, for instance) black, and the rest, white. That would create a mask, that can be blurred to use Hough Circles and detect the region of the pupil. Finally, I can set a radius for the iris.

Any idea would be appreciated.

Thanks.

python opencv image-processing

Share Edit Follow

asked Aug 4 '17 at 19:06

sosegon
99 1 1 8

tesnorflow image classification – Bender Bending Aug 4 '17 at 19:08

1 Are you just trying to classify the eyes into 'blue' or 'not blue'? If so, you could look at the colour histogram of the entire image and look for a spike of blue tones that don't otherwise appear in skin of any colour. You should be able to identify the presence of blue hues in the image, or the lack of blue. – struthersneil Aug 4 '17 at 19:13

what is your question? how many eye-colors do you want to distinguish? blue and brown? green blue brown? or even differend shades of brown? I'd go with @struthersneil advice. crop the center of these pictures and classify the hue . there should be a big difference between brown and blue eyes – Piglet Aug 4 '17 at 21:06

2 Answers

Active | Oldest | Votes

0

Actually your idea of detecting the shape of the pupil is good but your pictures are not good enough to do it directly. An easy way is to pre-process those to remove all useless data.

I did some example with one of your original pics to show you (on Gimp)

- Go to grey scale
- Do a high pass filter to remove all small color fluctuations (you have very distinct colors so it should enhance borders very well)

Link to example filtered pic

- Apply a threshold on your picture to remove remaining fluctuations (you can calculate the reference threshold value by analyzing your grey scale image color histogram)

[Link to example thresholded pic]

After those three steps you should have enough data to run your shape detection.

Share Edit Follow

edited Aug 5 '17 at 1:02

answered Aug 5 '17 at 0:40

hackela
303 1 8

Thanks for your answer. What are the values of the filters you used?. I tried to apply them as you suggested, but I don't get similar results. – sosegon Aug 8 '17 at 1:42

I'm using Gimp filters which are a bit more involved than standard high pass and threshold you can code but it gives you a good idea of the operation to do. Here are have params I extracted: For high pass: Normalize: 0% Hue: 0 Sauturation: 100% Brightness: 100% Gamma value: 1 Contrast: 0% Equalize: 0% For the threshold: I just put the contrast to 100% (only extreme values are left) – hackela Aug 9 '17 at 0:10

-1

Most of the answers I have read till now say to use the Hough circle method to detect the iris region, but it doesn't really work on all images.

So my approach is pretty simple, which involves following steps

- **Detect face from the image**
- **Find eye region from the face**
- Get the **RGB values** just below the pupil region(thereby getting the **iris region RGB** values)
- And pass the obtained RGB values to **find_color** function

NOTE: **Pass High-resolution image as the input for better results. If you pass low-resolution images such as 480x620, 320x240, you might end up getting poor results.**

Below is the code for the same

```
import cv2
import imutils
from imutils import face_utils
import dlib
import numpy as np
import webcolors


flag=0
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

img= cv2.imread('blue2.jpg')
img_rgb= cv2.cvtColor(img,cv2.COLOR_BGR2RGB)   #convert to RGB


#cap = cv2.VideoCapture(0)             #turns on the webcam

(left_Start, left_End) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
```

```
#points for left eye and right eye
(right_Start, right_End) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]



def find_color(requested_colour):            #finds the color name from RGB values

        min_colours = {}
        for name, key in webcolors.CSS3_HEX_TO_NAMES.items():
            r_c, g_c, b_c = webcolors.hex_to_rgb(name)
            rd = (r_c - requested_colour[0]) ** 2
            gd = (g_c - requested_colour[1]) ** 2
            bd = (b_c - requested_colour[2]) ** 2
            min_colours[(rd + gd + bd)] = key
            closest_name = min_colours[min(min_colours.keys())]
        return closest_name




#ret, frame=cap.read()
#frame = cv2.flip(frame, 1)

#cv2.imshow(winname='face',mat=frame)

gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)

# detect dlib face rectangles in the grayscale frame
dlib_faces = detector(gray, 0)


for face in dlib_faces:
    eyes = []                           # store 2 eyes

    # convert dlib rect to a bounding box
    (x,y,w,h) = face_utils.rect_to_bb(face)
    cv2.rectangle(img_rgb,(x,y),(x+w,y+h),(255,0,0),1)       #draws blue box over face



    shape = predictor(gray, face)
    shape = face_utils.shape_to_np(shape)

    leftEye = shape[left_Start:left_End]
    # indexes for left eye key points

    rightEye = shape[right_Start:right_End]

    eyes.append(leftEye)  # wrap in a list
    eyes.append(rightEye)

    for index, eye in enumerate(eyes):
        flag+=1
        left_side_eye = eye[0]  # left edge of eye
        right_side_eye = eye[3]  # right edge of eye
        top_side_eye = eye[1]  # top side of eye
        bottom_side_eye = eye[4]  # bottom side of eye

        # calculate height and width of dlib eye keypoints
        eye_width = right_side_eye[0] - left_side_eye[0]
        eye_height = bottom_side_eye[1] - top_side_eye[1]

        # create bounding box with buffer around keypoints
```

```
            eye_x1 = int(left_side_eye[0] - 0 * eye_width)
            eye_x2 = int(right_side_eye[0] + 0 * eye_width)

            eye_y1 = int(top_side_eye[1] - 1 * eye_height)
            eye_y2 = int(bottom_side_eye[1] + 0.75 * eye_height)

            # draw bounding box around eye roi

            #cv2.rectangle(img_rgb,(eye_x1, eye_y1), (eye_x2, eye_y2),(0,255,0),2)


            roi_eye = img_rgb[eye_y1:eye_y2 ,eye_x1:eye_x2]     #  desired EYE Region(RGB)
            if flag==1:
                break

    x=roi_eye.shape
    row=x[0]
    col=x[1]
    # this is the main part,
    # where you pick RGB values from the area just below pupil
    array1=roi_eye[row//2:(row//2)+1,int((col//3)+3):int((col//3))+6]

    array1=array1[0][2]
    array1=tuple(array1)   #store it in tuple and pass this tuple to "find_color" Funtion



    print(find_color(array1))







    cv2.imshow("frame",roi_eye)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Below are some examples.

- An actress with blue eyes

Now this is the output of our code when the above image is given as the input: **lightsteelblue**

- An actress with brown eyes

The output of our code when the above image is given as the input: **saddlebrown**

- Mila kunis (one brown eye and other is green)

The output of our code when the above image is given as the input: **sienna(shade of brown)**

- An actress with grey eyes

The output of our code when the above image is given as the input: **darkgrey**

So, you can see how close the results are to the actual eye color. This works pretty well with high-resolution images as I already mentioned.

PS: Correct me if am wrong, open to suggestions.

Share Edit Follow

answered Apr 25 at 8:57

Shrihari Kulkarni

1