

Assignment-1

Computer Vision (CSL7360)

Q1: Red Eye Detection

Abstract:

Red-eye, a prevalent issue in flash photography, manifests as the reddening of human pupils in captured images. To address this challenge, I have utilized the Python “Haar Cascade” algorithm to detect face and red-eye occurrences in images. Python program is developed for the automated identification of red-eye anomalies within images. used few sample images for evaluation purposes, demonstrating the algorithm's efficacy in mitigating red-eye effects.

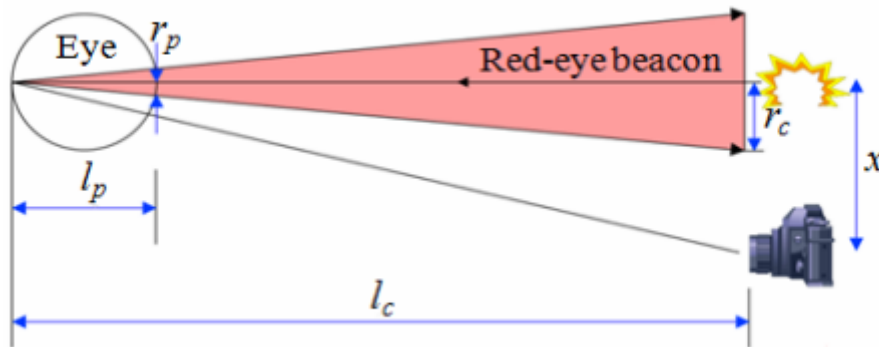
INTRODUCTION:

In scenarios where photography is undertaken under low-light conditions, the application of a flash is a common practice to enhance visibility. Nevertheless, the utilization of a flash often leads to the emergence of the notorious red-eye effect. This phenomenon stems from the flashlight passing through dilated pupils, subsequently reflecting off the blood vessels within the eyes, and ultimately reaching the camera lens. The outcome is the portrayal of red-toned eyes in the captured image. I've come up with a solution to tackle this using Python program. This program is designed to find red eye present in the given images and identify and mark the face and eye region.

Cause of Red-eye Effect:

Red-eye effect results from flashlight of a camera. When it is dark, pupils of eyes are enlarged. Through the enlarged pupils,

flashlight from a camera passes the eye lens and is reflected-back. This reflected light is blood-color since it is reflected from the blood vessels inside an eye. Fig. describes the geometric situation in which red-eye effect occurs. The light reflected from the eye composes the red-eye beacon. If a camera lens which receives the light is located inside the red-eye beacon, pupils in the picture look red. In other words, red-eye effect occurs if.



$$x < r_c / 2 \text{ -----(1)}$$

Where

x is the distance between a camera lens and flashlight, and r_c is the radius of the red-eye beacon at the camera position is shown in figure

$$r_c = r_p \times l_c / l_p \text{ ----- (2)}$$

where l_p , l_c , and r_p denote the diameter of eye sphere, the distance between an eye and a camera, and radius of pupil, respectively. From

equation (1) and (2), it is noted that red-eye effect can occur more frequently when x is small and l_c and r_p are large.

Steps Followed for Red Eye Detection:

To detect eyes in an image and draw bounding boxes around them, I followed the following steps

- Imported the OpenCV library.
- Download the `haarcascade_frontalface_default.xml` and `haarcascade_eye_tree_eyeglasses.xml` into the project folder.
- Read the input image using `cv2.imread()` in and converted to grayscale.
- Initiate the Haar cascade classifier objects `face_cascade = cv2.CascadeClassifier()` for face detection and `eye_cascade = cv2.CascadeClassifier` for eyes detection. Passed the path of the haar cascade xml files. I used the haar cascade file `haarcascade_frontalface_default.xml` to detect faces in the image and `haarcascade_eye_tree_eyeglasses.xml` to detect eyes.
- Detect faces in the input image using `face_cascade.detectMultiScale()`. It returns the coordinates of detected faces in (x,y,w,h) format.
- Define roi as `image[y:y+h, x:x+w]` for the detected face. Now detect eyes within the detected face area (roi). Use `eye_cascade.detectMultiScale()`. It also returns the coordinate of the bounding rectangle of eyes in (ex,ey,ew,eh) format.
- Draw the bounding rectangles around the detected eyes in the original image using `cv2.rectangle()`.
- Display the image with the drawn bounding rectangles around the eyes.
- Based the threshold value of red color inside the eye boundaries red color present or not identified.

Results:

Case 1: Red Eye Not Present

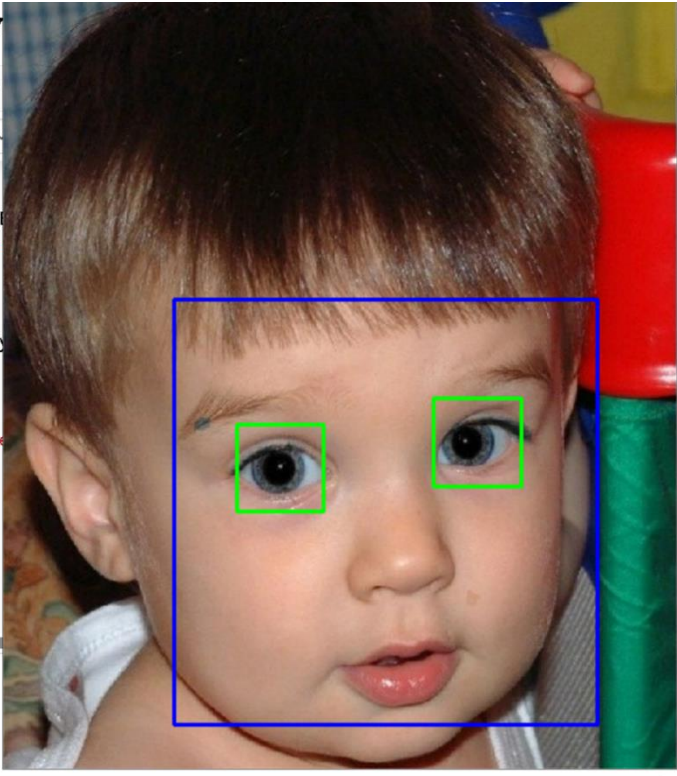
Jupyter RedEyeDetectionFinalversion_M22AI567

File Edit View Insert Cell Kernel Widgets Help

Code

```
55 #read image
56 img = cv2.imread("noredeye.jpg")
57 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
58 faces, faces_present, total_faces = detect_faces(gray)
59 print(total_faces)
60 if faces_present:
61     for face in faces:
62         eyes, total_eyes = detect_eyes(face)
63         print('eyes count',total_eyes)
64
65 # Display image
66 cv2.imshow('Red Eye marked Image',img)
67 cv2.waitKey(0)
68 cv2.destroyAllWindows()
69 else:
70     print('No Face detected')
71
72
73
```

1
No red-eye detected in the eye region.
No red-eye detected in the eye region.
eyes count 2



Case 2: Red Eye Present

Jupyter RedEyeDetectionFinalversion_M22AI567

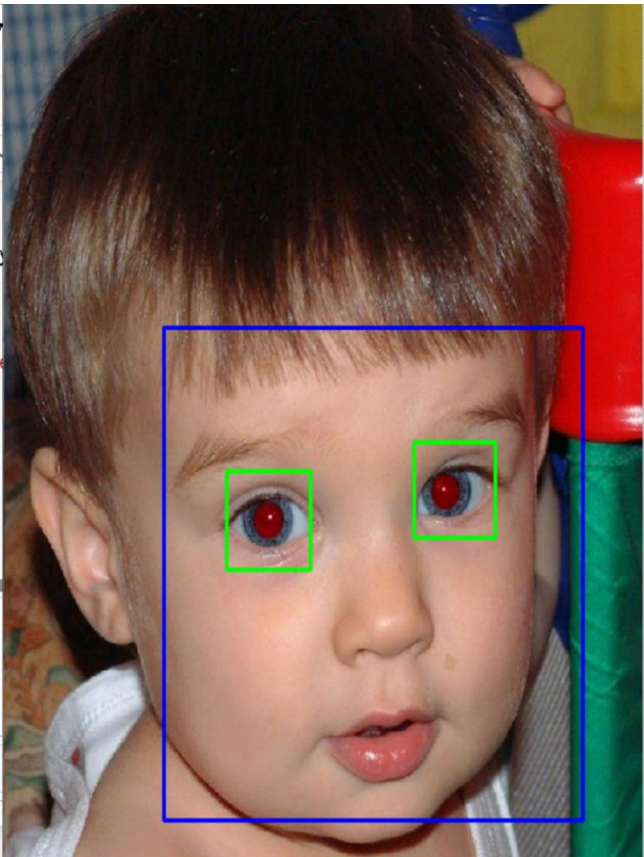
File Edit View Insert Cell Kernel Widgets Help

Code

```
61 if faces_present:
62     for face in faces:
63         eyes, total_eyes = detect_eyes(face)
64         print('eyes count',total_eyes)
65
66 # Display image
67 cv2.imshow('Red Eye marked Image',img)
68 cv2.waitKey(0)
69 cv2.destroyAllWindows()
70 else:
71     print('No Face detected')
72
73
74
```

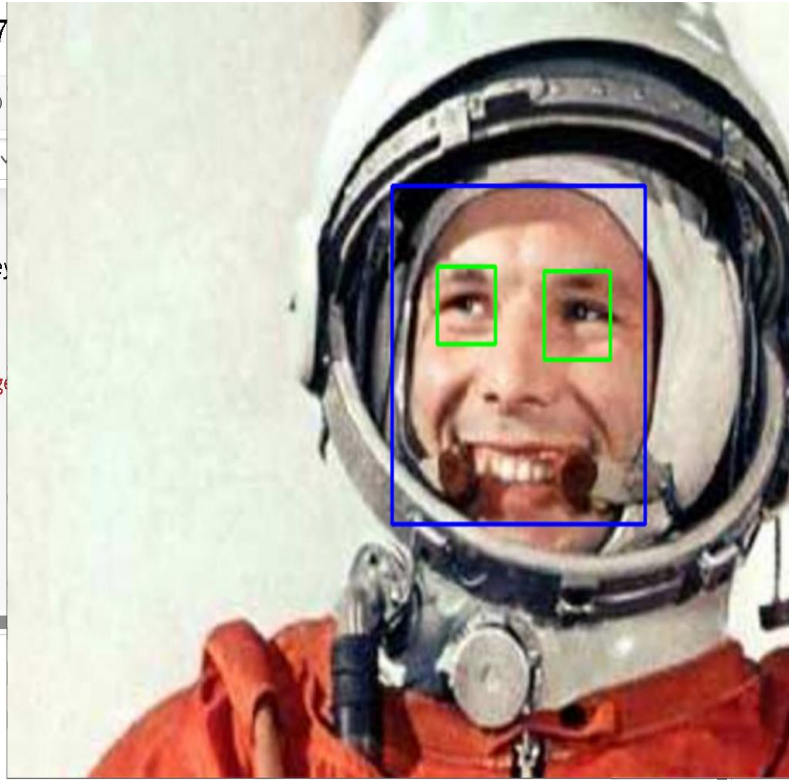
1
Red-eye detected in the eye region.
Red-eye detected in the eye region.
eyes count 2

In []: 1




```
61 if faces_present:
62     for face in faces:
63         eyes, total_eyes = detect_e
64         print('eyes count',total_eyes)
65
66     # Display image
67     cv2.imshow('Red Eye marked Image
68     cv2.waitKey(0)
69     cv2.destroyAllWindows()
70 else:
71     print('No Face detected')
72
73
74
```

```
1
Red-eye detected in the eye region.
Red-eye detected in the eye region.
eyes count 2
```



Case 3: No Face in the Image

```
64 print('eyes cou
65
66 # Display image
67 cv2.imshow('Red
68 cv2.waitKey(0)
69 cv2.destroyAll
70 else:
71     print('No Face
72     cv2.imshow('No
73     cv2.waitKey(0)
74     cv2.destroyAll
75
76
77
```

```
0
No Face detected
```



REFERENCES:

- S. Ioffe, "Red eye detection with machine learning," Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429), Barcelona, Spain, 2003, pp. II-871, doi: 10.1109/ICIP.2003.1246819.
- M. Gaubatz and R. Ulichney, "Automatic red-eye detection and correction," *Proceedings. International Conference on Image Processing*, Rochester, NY, USA, 2002, pp. I-I, doi: 10.1109/ICIP.2002.1038147.
- Red Eye Correction from an Image using OpenCV.
<https://pytech-solution.blogspot.com/2017/04/red-eye-correction-from-image-using.html>
- How to detect eyes in an image using OpenCV Python?
<https://www.tutorialspoint.com/how-to-detect-eyes-in-an-image-using-opencv-python>

Q2 skin detection

Abstract

Human Skin detection deals with the recognition of skin-colored pixels and regions in a given image. Skin color is often used in human skin detection because it is invariant to orientation and size and is fast to process. The three main parameters for recognizing a skin pixel are RGB (Red, Green, Blue), HSV (Hue, Saturation, Value) and YCbCr (Luminance, Chrominance) color models. The objective of this problem is to improve the recognition of skin pixels in given images.

INTRODUCTION:

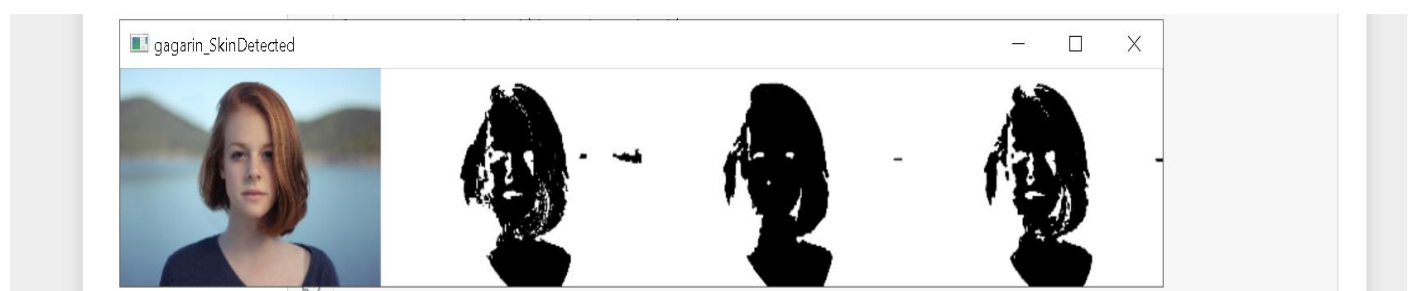
Skin detection is the process of finding skin-colored pixels and regions in an image or a video. This process is typically used as a preprocessing step to find regions that potentially have human faces and limbs in images. Skin image recognition is used in a wide range of image processing applications like face

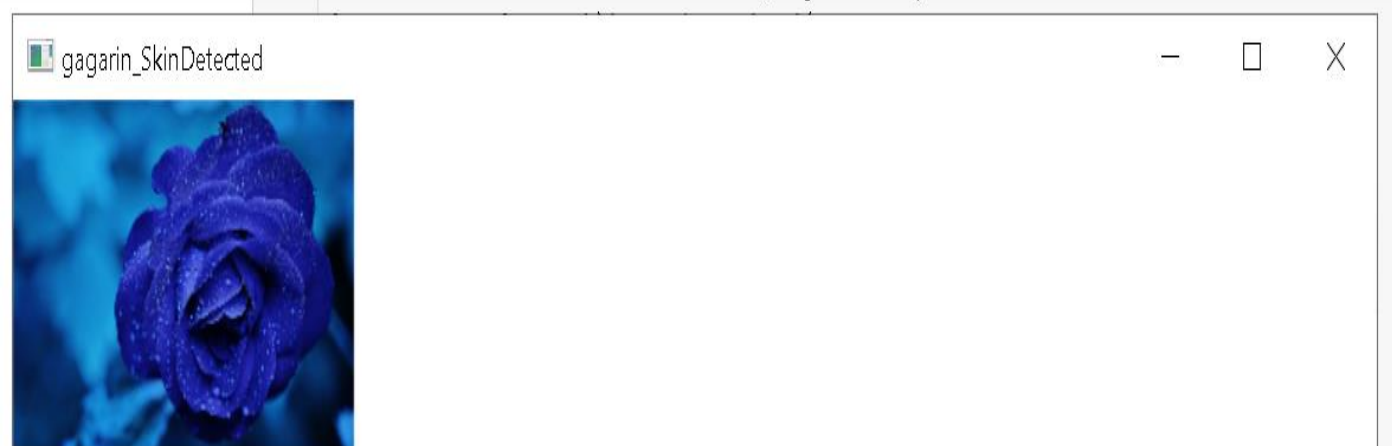
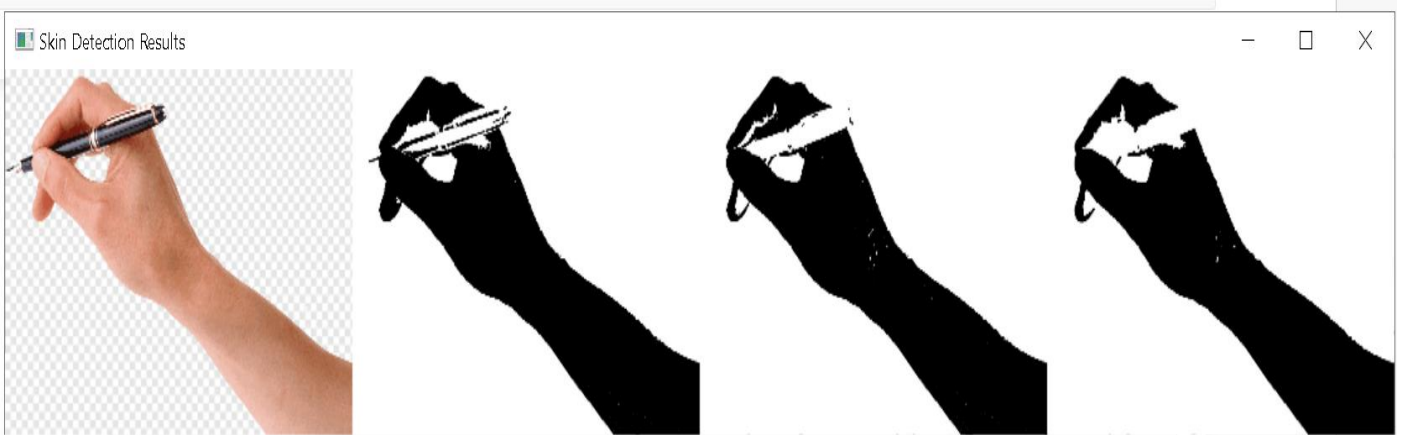
recognition, skin disease detection, gesture tracking and human-computer interaction. The primary key for skin recognition from an image is the skin color. But color cannot be the only deciding factor due to the variation in skin tone according to different races. Other factors such as the light conditions also affect the results. Therefore, the skin tone is often combined with other cues like texture and edge features. This is achieved by breaking down the image into individual pixels and classifying them into skin colored and non-skin colored. One simple method is to check if each skin pixel falls into a defined color range or values in some coordinates of a color space. There are many skin color spaces like RGB, HSV, YCbCr, YIQ, YUV, etc. that are used for skin color segmentation.

Steps Followed for Skin Detection:

- Importing Required Libraries
- Loading and Processing the Image
- Converting to HSV Color Space and Applying Mask
- Converting to YCbCr Color Space and Applying Mask
- Combine Skin Detection from Different Color Spaces

Results:





Summary

Skin detection in color images and videos is a very efficient way to locate skin-colored pixels, which might indicate the existence of human faces and hands. However, many objects in the real world have skin-tone colors, such as some kinds of leather, sand, wood, fur, etc., which might be mistakenly detected by a skin detector. Therefore, skin detection can be very useful in finding human faces and hands in controlled environments where the background is guaranteed not to contain skin-tone colors. Since skin detection depends on locating skin-colored pixels, its use is limited to color images, i.e., it is not useful with gray-scale, infrared, or other types of image modalities that do not contain color information. There have been extensive research on finding human faces in images and videos using other cues such as finding local facial features or finding holistic facial templates. Skin detection can also be used as an efficient preprocessing filter to find potential skin regions in color images prior to applying more computationally expensive face or hand detectors.

References:

- Q. X. Wu, R. Cai, L. Fan, C. Ruan and G. Leng, "Skin detection using color processing mechanism inspired by the visual system," IET Conference on Image Processing (IPR 2012), London, 2012, pp. 1-5, doi: 10.1049/cp.2012.0459.
- Skin Detection - a Short Tutorial, Ahmed Elgammal, Crystal Muang and Dunxu Hu Department of Computer Science, Rutgers University, Piscataway, NJ, 08902, USA
- Human Skin Color Detection Using Neural Networks
<https://www.degruyter.com/document/doi/10.1515/jisys-2014-0098/html?lang=en>
- Human Skin Detection Using RGB, HSV and YCbCr Color Models
<https://arxiv.org/ftp/arxiv/papers/1708/1708.02694.pdf>