Assignment 3

CP467: Image Processing & Recognition
Professor: Dr. Zia Ud Din
Due Date: Tuesday October 31, 2023. 11:59 p.m.

Chandler Mayberry 190688910

Table of Contents

Introduction

All the code for this assignment may be run using the a03_190688910.py file in the source code folder. Running this file will perform the operations and display the resulting images to the user using python OpenCV.imshow() method.

Circle Detection Code Explanation:

The purpose of this assignment is to determine the circular locations of the iris and the pupil within images of eyes. My approach to this task is relatively straight forward but does differ when determining the iris circle over the pupil circle. To begin, the main algorithm I decided to use to extract the location of the circles was Hough transform from the OpenCV library (*OpenCV: Feature Detection*). This algorithm as discussed in lecture is robust and more tolerant to noise (which is the main advantage over something like RANSAC), with the advantage of being able to determine other geometric primitives in an image such as circles by default.

To find the location of the pupil in the image was rather simple, the pupil itself has high contrast and is a near perfect circular shape in every instance. The only preprocessing I perform on the image before running cv2.HoughCircles is a blur function to help reduce the amount of noise in the image (performing the operation without blur caused the pupil not to be found in some of the eyes when testing). The blur function I chose to use on each of the images for both operations was cv2.medianBlur, this is because it takes the median of a set of pixels and replaces the central element with the median value. This is particularly useful in this use case as no new value is created to replace each pixel (*OpenCV: Smoothing Images***)**. This helps reduce the number of sudden changes of pixel values in locations around the eye lashes and overall smooths the image to help differentiate edges from noise. In practice, using cv2.blur and cv2.gaussian blur ended up in incorrect results when running the pupil Hough circles operation, causing all circles to be incorrectly placed regardless of parameters used.

Moving forward, to determine the location of the iris in the image took more fine tuning and preprocessing. In practice, the Hough built-in method takes advantage of an edge map when extracting geometry from an image. As such, the default used by the built-in Hough method is Canny edge detection as can be read in the documentation (*OpenCV: Feature Detection*). To prepare the image for the edge map in both instances, I needed to reduce the amount of noise and detail in the image so Canny could find the most important edges (*OpenCV: Canny Edge Detection*). The detail that is the most troublesome is the eye lashes in the images. The only way I could think of how to remove this detail was the opposite way I wanted to exaggerate detail using cv2.dilation in the previous assignment to better demonstrate my grouping function (*OpenCV: Eroding and Dilating*). As such, I found the sister method in the OpenCV documentation called cv2.erode which computes the local minimum over a given area and essentially removes a lot of the detail in the image (*OpenCV: Eroding and Dilating*). After

blurring and applying erosion to the image, I then apply cv2.dilate to re-exaggerate the remaining lines of detail in the image (but with removed detail from erosion).

Finally, before running the Hough built-in for the iris circle I wanted to increase the contrast and brightness of the image to help in edge detection. The method to do this was the cv2.convertScaleAbs which simply scales the pixel values and applies a delta (I chose a scaling factor of 1.8 with a delta of -70) (*OpenCV: Operations on Arrays*). This removes the rest of the background detail and amplifies the edges in the eye, including the iris. After this, I run the HoughCircles method to find the iris circle, and thereafter place the circles produced from the iris and pupil back onto the original image. Then in the return function send back the Canny edge detection (demonstrating the one used by the built-in Hough method) and the image processed eye.

In conclusion, I believe there is a more effective and efficient way of preprocessing the image to help the Hough algorithm determine the placement of the iris circle. I notice now while writing the report, that I believe the reason my iris placement is a little larger than the actual iris in the image is because of the increase in size the iris undertakes in the erosion and dilation steps. Finding a better solution for reducing detail in the image in finding the iris would be my greatest future improvement.

Preprocessing Step Examples:

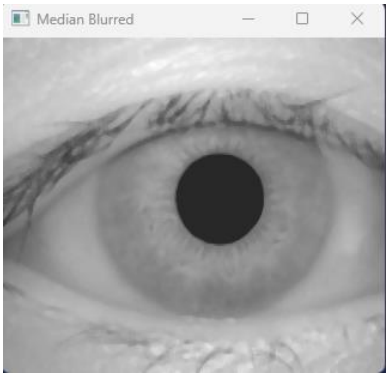

Figure 1: Preprocessing Eyeball 1 Median Blur

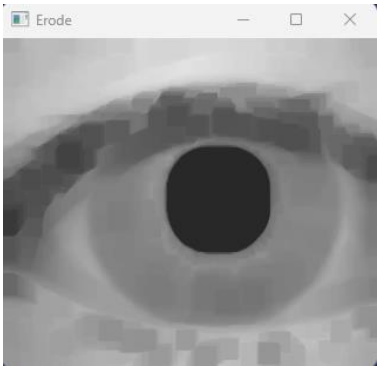

Figure 2: Preprocessing Eyeball 1 Erode Image

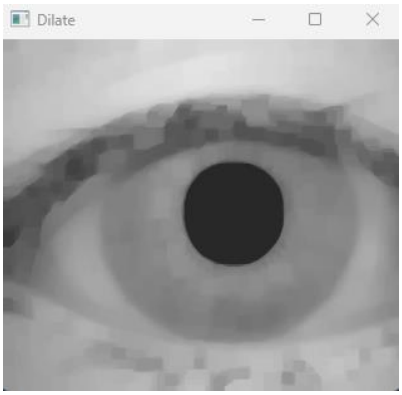

Figure 3: Preprocessing Eyeball 1 Dilate Image

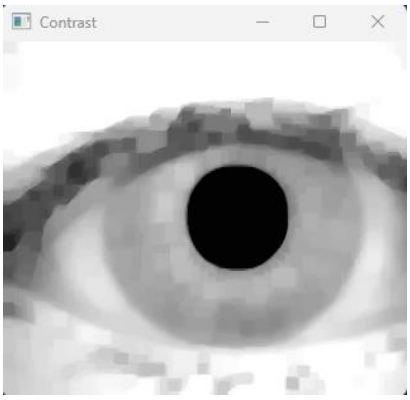

Figure 4: Preprocessing Eyeball 1 Contrast Increase

Edge Map & Circle Detection Results

Figure 5: Eyeball 1 Canny Edge Map

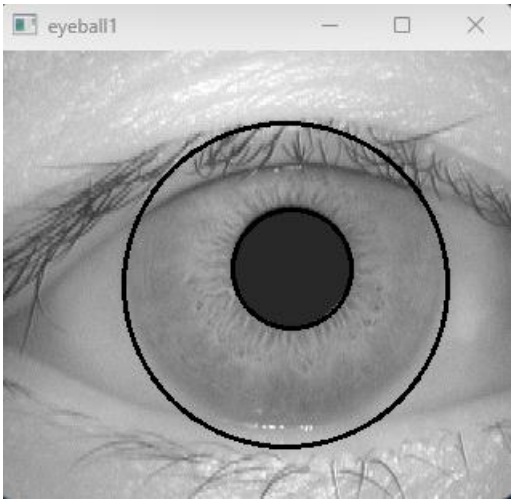

Figure 6: Eyeball 1 Hough Circle Detection

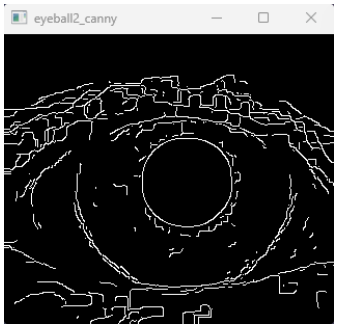

Figure 7: Eyeball 2 Canny Edge Map

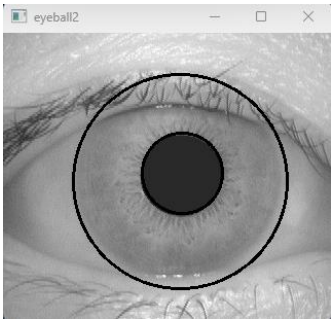

Figure 8: Eyeball 2 Hough Circle Detection

Figure 9: Eyeball 3 Canny Edge Map

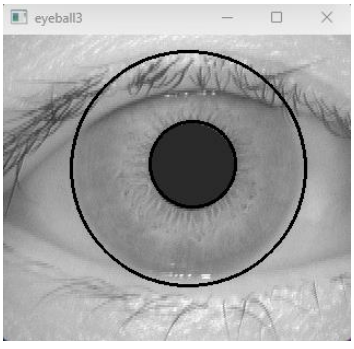

Figure 10: Eyeball 3 Hough Circle Detection

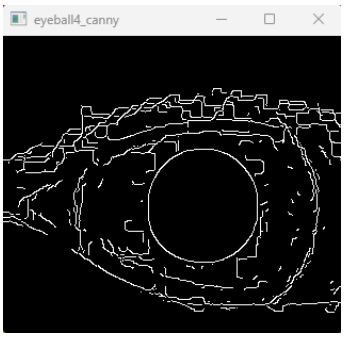

Figure 11: Eyeball 4 Canny Edge Map

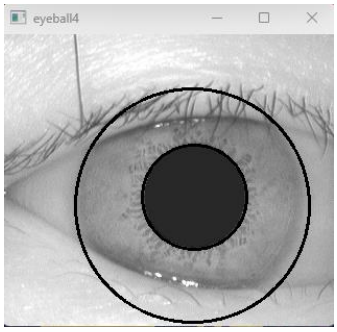

Figure 12: Eyeball 4 Hough Circle Detection

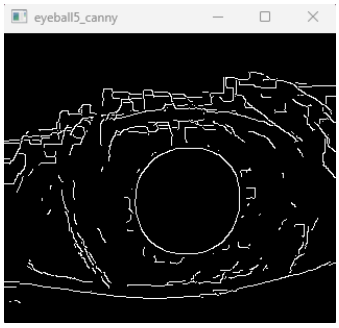

Figure 13: Eyeball 5 Canny Edge Map

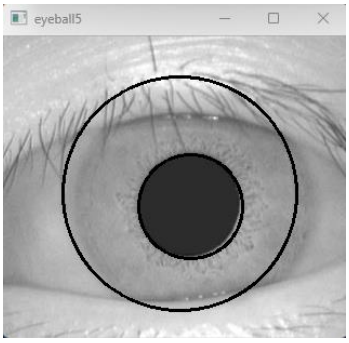

Figure 14: Eyeball 5 Hough Circle Detection

Works Cited

OpenCV: Canny Edge Detection. docs.opencv.org/4.x/da/d22/tutorial_py_canny.html.

OpenCV: Eroding and Dilating. docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html.

OpenCV: Feature Detection. docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html.

OpenCV: Operations on Arrays. docs.opencv.org/3.4/d2/de8/group__core__array.html.

OpenCV: Smoothing Images. docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html.