Eye-Tracking Painter

COMP 4102 - Project Deliverable Paper
Michael Simpson
101074874
April 9th, 2022

Abstract

The algorithm that I developed will detect where a subject is looking and proceed to draw a point on a canvas, effectively allowing the subject to draw using only eye movements. The Haar Cascade classifier allowed for easy facial and eye detection, where I then developed my own algorithm using blurs, thresholding and contours to detect the position of the center of the pupil in relation to the center of the eye. With this relation, I created a canvas where the point will be drawn.

Introduction

I chose this as my topic for my project because I have always been intrigued by software and devices that would track where a person would be looking on the screen. This always seemed like something that was completely out of reach for my knowledge-set, that is, until I took this class. This class taught me a lot about the different techniques that would be used in the commercial-grade eye tracking software, which brought my thoughts immediately to how I would implement them. Thus, this project was thought out, with the change being to paint on a canvas instead of tracking location on your monitor.

At first, this seemed like a pretty straight-forward task. Unbeknownst to me however, I would run into many sets of challenges that I would need to overcome. Challenges such as:

1. tkinter canvas running the mainloop and thus not allowing for any code to run afterward.
2. The eye_haarcascade tries to centralize on the pupil/iris, causing no movement to be tracked.
3. Actually finding where the pupil is, and the algorithm that I needed to implement using contours.The eye tracker tracking both eyes (and other “eyes” that were false positives)

Background

There are two important computer vision topics that I made use of during this project. The first being Haar Cascade classifiers and the second being contours.

A Haar Cascade is a machine-learned algorithm developed by Paul Viola and Michael Jones in 2001 (Shetty *et al*, 2021). It has an extremely high facial-detection rate, and works by comparing features within an image to the features that a specific object may have, or in my case, facial features (Singh *et al*, 2013). There are two classifiers that I will make use of during this project: the facial classifier and the eye classifier, and what the cascade algorithm will do is

first scan the image to find different facial features, returning a rectangular frame, and then it will scan that rectangular frame to find eyes (Phase, 2019).

Contours are the second major component to this project, and will be used when detecting the pupil within the image. A contour is essentially an edge-tracking algorithm which will traverse the border of a region in a binary image, eventually returning the edge points of that region (Xie, 2013). After using a binary threshold, using the OpenCV function cv2.FindContours(...) will return the edges of each individual white region within an image - making contours perfect for the isolation of the pupil within an image.

These topics are essential for my algorithm to work, as they implement extremely large algorithms to handle how my algorithm will narrow down on a subject's eyes. Without them, I would need to write my own complex machine-learning code to find faces within an image, and also vastly improve my own edge-detection function that I created in assignment 1 of this course.

Methodology

To begin, I first broke this up into many smaller and more obtainable tasks, with the end goal being finding the location of the pupil. My working knowledge of how HaarCascades work helped greatly in breaking down to a finite set of steps, of which I will be going into detail as to how they were accomplished:

First, detect a face in the image

This was accomplished using a basic Haar Cascade function, which will return the location of the top-left-most point of a face, followed by the length and width of the corresponding box around the face.

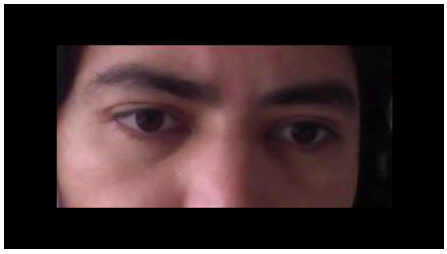

Figure 1: With this completed, I tried testing this with the eye gaze dataset, as mentioned in my proposal. Sadly, the Haar cascade was not able to detect a full face within these images, assumingly because the classifier also needs to be able to view the nose and mouth of a subject.

Second, detect the eyes in an image

This was also accomplished using the Haar Cascade, but only on the region of interest found by the previous use of the haar cascade for finding faces. When using this eye-detection algorithm, I was running into false positives where a subject's mouth was being detected as an eye… I resolved this issue by modifying the region of interest to only include half of the height (which will include the tip of the nose plus eyes and forehead). With this change, I was getting a much more accurate detection of the eyes within a frame.

Third, detect the pupil in an image

This was the beginning of my own algorithm, as stated in my proposal. I used a variety of methods as taught in class to be able to accomplish this. To begin, since the pupil is generally very dark compared to the iris I used an inverse binary threshold that will detect values lighter than X. As a proof of concept, I first ran this threshold on the entire face capture, just to see what type of results I would receive.

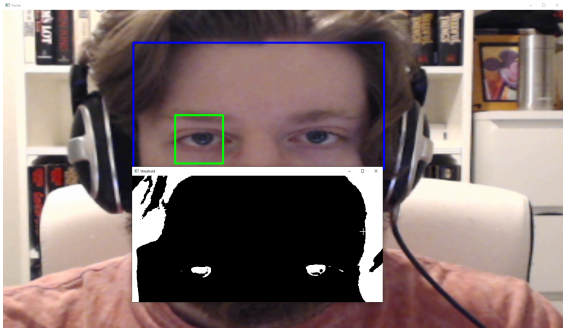

Figure 2: The iris and pupils of the eyes are being thresholded very well which was extremely encouraging.

I ran this function on just the detected eyes and came across the problem of a window opening for each eye that was being detected (also false-positive eyes). A solution that I came up with for this issue was to only take into account the eye with the largest area. This eliminated any false-positives and will come to decrease any confusion about which point to draw on the canvas down the line.

After running this function on just the eyes, I was then faced with the issue of actually getting the location of the detected blobs on the screen - which is where contours came into play. Using the built-in OpenCV contours functions I was able to detect the blobs and draw around the perimeter of them. I once again only used the contour with the largest area to ensure that no outliers found during the thresholding would be mistaken for a pupil. To further how useful these contours were, I was able to get the center point of the blobs with much ease.

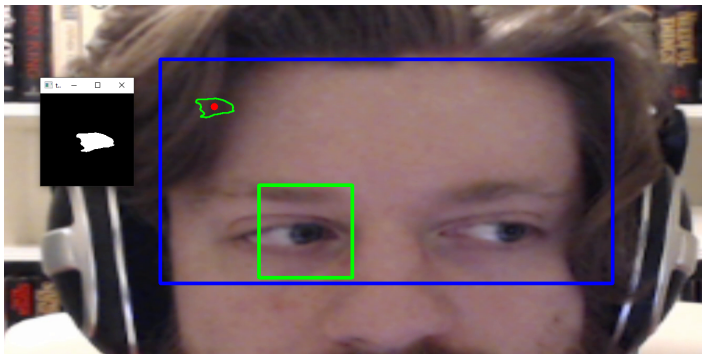

Figure 3: This picture displays the window with the thresholded image of the eye, plus the green contour (purposefully drawn away from the eye to demonstrate the shape), and finally the red dot which symbolizes the middle of the contour. The red dot also symbolizes the middle of the pupil.

With this center point, I have successfully detected the center of the pupil (with a relatively high confidence rate). Something I noticed during my testing is that the algorithm has a preference for those with darker eyes, as it makes it easier for the threshold to detect the center of the pupil/iris.

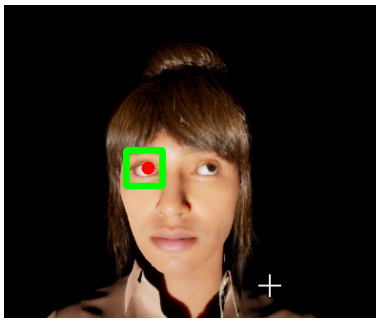

Figure 4: This is another of the datasets that I used, as mentioned in the proposal: synthetic gaze and face segmentation. I created a video with ~150 of the images and ran it through the algorithm multiple times to test.

Fifth, find the location of the pupil in relation to the origin

In the previous step, I was able to find the center point of the eye using contours, and with this point I am able to calculate the position of where the eye is currently looking. The reference area I will be using is the region of interest as generated by the eye_cascade, as that will give a fairly good estimation of the position of the point. In theory, this region can actually be referred to as the canvas because I will be drawing the points in a way that will directly correspond to the position of the red point in relation to the green square.

To be able to properly scale this to any canvas size, I calculate the position as a percentage about the origin (0,0), and can then use these percentages by multiplying them with the canvas height and width to get the proper position in relation to the canvas size.

As stated in the proposal, I also created a debugging option for when running the program which will calculate the angle between the pupil location and the middle of the image (the origin).

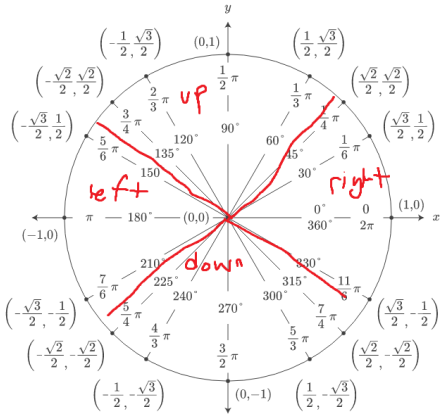

Figure 5: This is how the direction of the eye is translated into Up, Down, Left, Right. If the angle falls between a specific set of values, then the corresponding direction will be printed to the console.

Sixth, create a canvas and draw a point

Using the Python graphical framework tkinter, I immediately ran into an issue with the way that canvas' are commonly run. They use a mainloop() function that will effectively take over a program until that mainloop() closes, and when I added this to my program I immediately saw that the canvas would be drawn, but the algorithm that I wrote would only run after I closed this. I needed to be able to run these at the same time. I eventually came across a function that can be used to manually call updates to the canvas - namely: update(). When I call this at the end of my algorithm, the canvas will redraw with all points previously added.

To finish off my algorithm, I add a draw_rectangle() function which will draw a point at the position I calculated at the previous step, using the height and width of the canvas.

Results

With everything completed, I am very proud of the outcome. When initially running the program, you will be greeted with a few options: if you would like debug mode on, and if you would either like to run the algorithm with a video or your webcam. In the zip folder containing the program I have supplied 3 videos. One of the videos I created using the synthetic gazes algorithm and the other two are recordings I took that demonstrate how the algorithm runs through a webcam.

```
PS C:\Users\Mike\Desktop\Computer Vision\Project> python3 .\project.py
Welcome to the Eye-Tracking Painter application.
NOTE: Pressing 'q' will terminate the program.

To use debug mode (to see angle of points), type '1'
otherwise, type '0'
Enter a value: 1
To use the supplied input named 'SyntheticGazesDataset.mp4', type '1'
To use the supplied input named 'WebcamDemonstration.mp4', type '2'
To capture input from your webcam, type '3'
```

Figure 6: The text you will be greeted with when you run the program in the console.

Two windows will open, one where the drawing is happening and the second is of the input data of the algorithm.

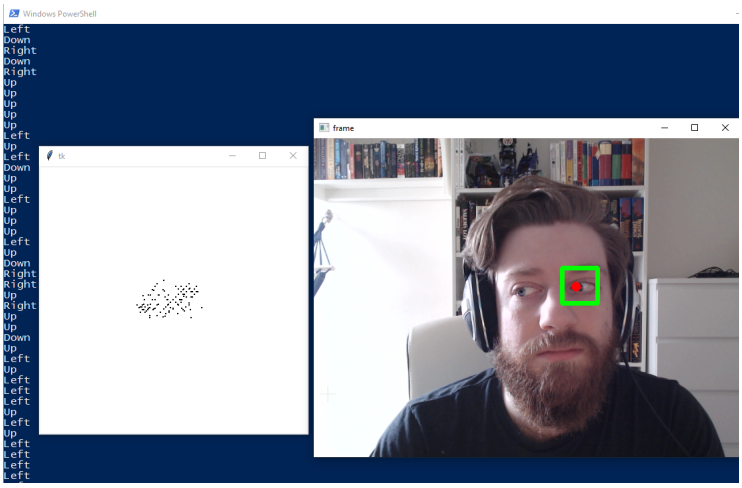

Figure 7: The background is the console in which I ran the algorithm, the white window on the left is the canvas where the drawing is happening, and finally the right is the input data.

Discussion

My work is relevant because it takes the concepts of thresholding, gaussian blurs, points of interest and contours as learned in class and applies them in a single algorithm.

There are a few limitations that my project has including:

- Preference to subjects with darker eyes (as described above).
- Cannot be wearing glasses. Glare and reflections may mess up pupil detection.
- Must be close up to the camera to more accurately draw on the canvas

As for future work, currently the algorithm will only draw in the shape of the eye on the canvas (see image below). A solution that I have come up with is to reduce the height of the eye detection to more accurately depict the eye socket instead of just the eye. This will scale looking around with the algorithm to be able to use the entire canvas, instead of just the center.

Another item that would be great to address fully is the fact that the eye_haarcascade attempts to focus the region of interest around the iris/pupil of the eye. This adds an amount of inaccuracy to my algorithm, and I believe this can be solved by taking a non-maximum suppression of the regions across multiple frames and only changing the size and location of a frame if, from one frame to the next, there is a different position and direction above a certain percentage threshold. This would help the algorithm become much more accurate in representing where the subject is looking due to removing the way that the haar cascade likes to center the region of interest for every frame upon the pupil.

References

Phase, M. T. (2019). Face detection and eye detection in image and video using pre-trained Haar-Cascade classifier. *International Journal for Research in Applied Science and Engineering Technology*, *7*(11), 652–659. https://doi.org/10.22214/ijraset.2019.11104

Shetty, A. B., Bhoomika, Deeksha, Rebeiro, J., & Ramyashree. (2021). Facial recognition using haar cascade and LBP classifiers. *Global Transitions Proceedings*, *2*(2), 330–335. https://doi.org/10.1016/j.gltp.2021.08.044

Singh, V., Shokeen, V., & Singh, B. (2013). FACE DETECTION BY HAAR CASCADE CLASSIFIER WITH SIMPLE AND COMPLEX BACKGROUNDS IMAGES USING OPENCV IMPLEMENTATION, *1*(12).

Xie, G., & Lu, W. (2013). Image edge detection based on Opencv. *International Journal of Electronics and Electrical Engineering*, *1*(2), 104–106. https://doi.org/10.12720/ijeee.1.2.104-106