EECS 225B Digital Image Processing
2022 Fall Class Project Report

Eye-Tracking System Under Different Lighting Conditions

Berkeley EECS
Chin-An (Daniel) Chen
Master of Engineering

Abstract

In this project, I am planning to implement an eye-tracking system that could detect the user's eyes in real-time under both normal and dim light environments.

Under normal light conditions, the overall pipeline would be first localizing the face within the image, then detecting the eyes within the face, and pointing out the center of the pupil in the end.

To further improve the system to support low-light conditions, I planned to implement a low-light enhancement module to be added right before doing the face localization, and the following modules are the same as the system under normal conditions.

Because this system targets detecting the user's eyes correctly in real-time, I would emphasize the accuracy and the runtime to compare different algorithms.

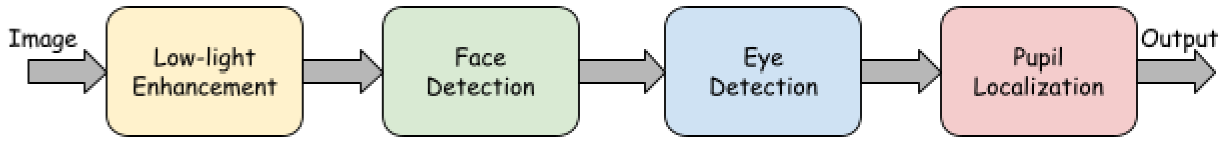
Backgrounds

I am working on the Vision Correcting Display(VCD) as my capstone project under the supervision of Professor Brain A. Barsky. By VCD, we aim to provide an image that will be seen in sharp focus by a viewer without the use of eyeglasses or contact lenses; that is, the text or image on display is modified according to the measured vision problem of the viewer so that the transformed image will appear in sharp focus by the viewer. This could impact computer monitors, laptops, tablets, and mobile phones.

To realize the VCD system, the first step is to find the user's eyes, as well as the gaze angles, for the following processes to render the re-focused display for the users. Hence, I decided to implement the real-time eye-tracking system using low-light enhancement models we learned in this class to improve the VCD system to be supportive under different lighting conditions.

Algorithms

Here, I divided the overall datapath into four stages shown below to realize an eye-tracking system.



For each stage, there are several algorithms to satisfy the functionality. My plans would then be first to implement each of the algorithms, and connect their inputs and outputs into the whole system; afterward, compare their performances, such as runtime, accuracy for face and eye detection, and how precise they could point out the locations of the pupils, and decide which combinations of eye-tracking systems would be the most beneficial to the VCD system.
Below, I listed out all the candidates for each stage and the references.

- ❖ Low-Light Enhancement
  - Zero-Reference Deep Curve Estimation [8]
  - EnlightenGAN [9] (Not be implemented in this project)
  - Dual Illumination Estimation [5]
  - LIME [6]

- ❖ Face Detection
  - Haar Cascade [1]
  - HOG frontal face detector with SVM [11]
  - MTCNN [2]
  - DNN frontal face detector [12]

- ❖ Eye Detection
  - 68 key points facial landmarks [13]

- ❖ Pupil Localization [4]
  1. Get the images of eyes
  2. Brighten the Iris and Pupil
  3. Binarization to isolate the pupil regions
  4. Contour the pupils
  5. Localize the centers of the pupils

Face Detection

There are a bunch of methods to detect faces within one image; in this project, I found four ways to do face detection, including Haar Cascade, HOG frontal face detector with SVM, MTCNN, and DNN frontal face detector. Below, I would briefly introduce each of them and also implement four of them into my eye-tracking system. To be consistent, I made all of the face detectors take 320x240 images as input and would output the bounding boxes of faces within the image.

1. Haar Cascade

This method is proposed in [1], and the detection could be divided into three phases. The first phase is called Integral Image, which allows the features used by the detector to be computed very quickly. The following phase is using Adaboost as a learning algorithm to select a small number of important visual features among a larger set, through this, it could yield extremely efficient classifiers. The last phase is to use cascading by combining more complex classifiers which allows background regions to be discarded quickly and spending more computation on the face-like regions.

For the implementation, thankfully, there is a trained Haar Cascade model within the OpenCV library that I could use as a part of my system.

2. HOG Frontal Face Detector with SVM

In this method, the first step is to extract the HOG features out of the input grayscale image, and the second stage would then be passing the extracted features to the trained support vector machine to detect whether there are faces or not. The problem with this detector is that the HOG-based detector doesn't work well with odd angles, which means that the user needs to pose the straight-frontal face to the system to be detected.

For the implementation, I resorted to Dlib, which is a C++ toolkit that contains different machine-learning algorithms.

3. MTCNN

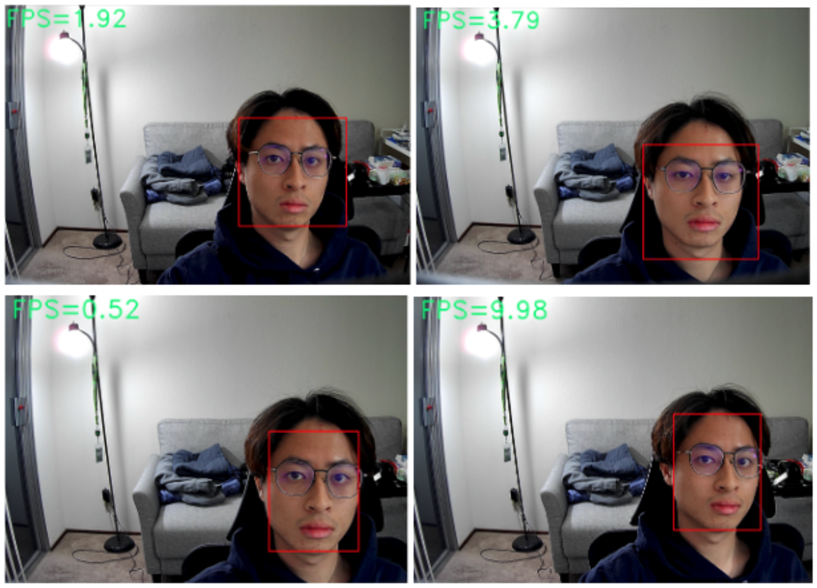
MTCNN was proposed in [2], it uses a cascade structure with three stages of CNN to detect the face in a coarse-to-fine manner. The first stage is to obtain candidate windows using CNN as well as their bounding boxes regression vectors, and using non-maximum suppression to deal with highly overlapped candidates; the second stage is to reject a large number of false positives and do the calibration of the bounding boxes with another CNN; the final stage would then be outputting the facial landmark detection.

For the implementation, I used the Python MTCNN library to implement it.

4. DNN Frontal Face Detector

It used ResNet-10 as the model architecture to build a single-shot multibox face detector. To implement it, I resorted to the OpenCV library, which includes the trained DNN face detector model within it.

[Results of Face detection] image size = 1080x1920x3
(Face detected is bounded by red rectangle)



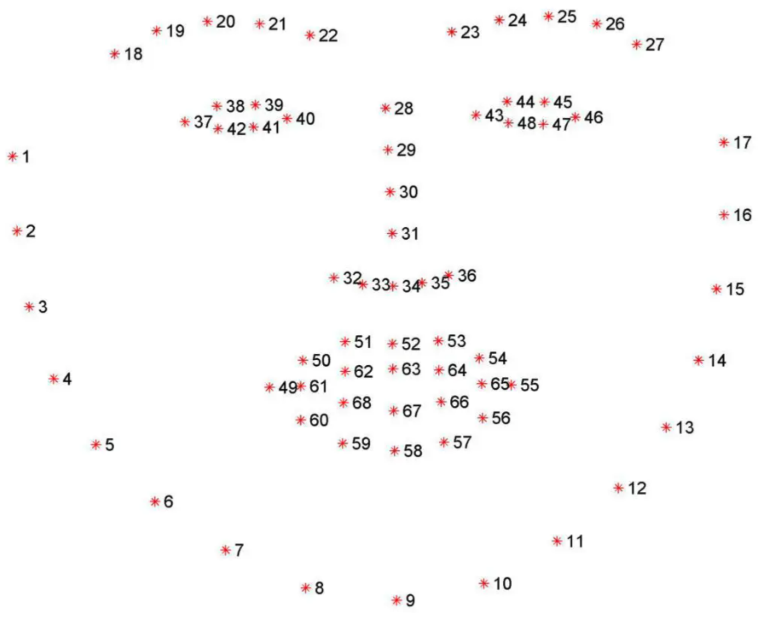
[Table 1.] Face-Detector Comparison (FPS)

Method	HAAR Cascade	HOG	MTCNN	DNN
FPS	1.92	3.79	0.52	9.98

Eye Detection

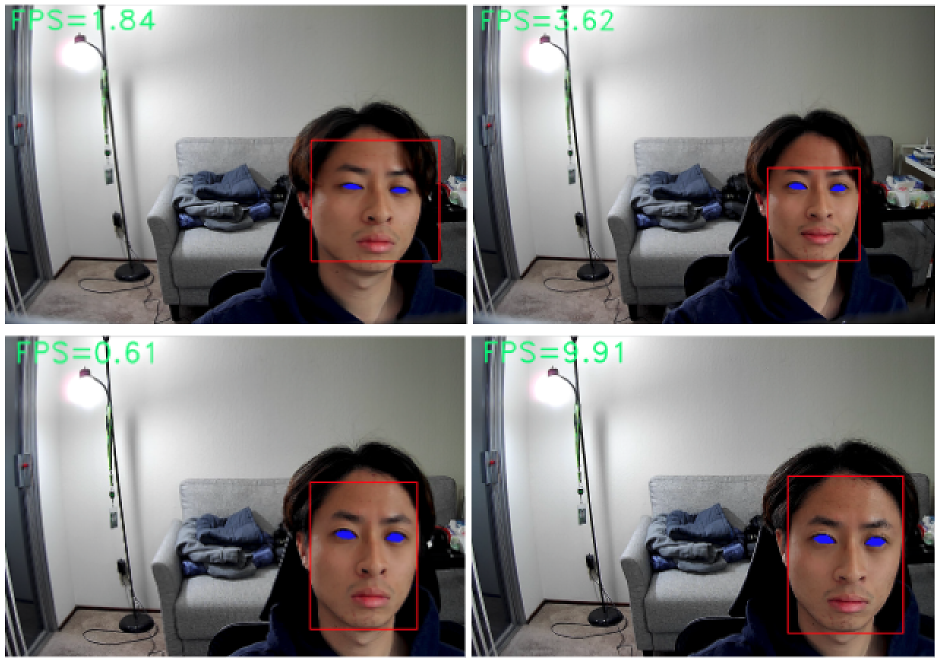
After localizing the bounding box of faces within the image, the next step is to find eyes within the faces. In this project, I used a 68-point facial landmark as the eye detector. Below, I introduced the concept and algorithm of facial landmark detection and implemented it within the eye-tracking system.

Based on [13], with the given face bounding boxes, a 68-facial landmarks detector would detect key facial structures within those bounding boxes, which contain mouth, right and left eyebrows, right and left eyes, nose, and jaw, for my eye-tracking system, only right and left eyes are in interested. (points 37-46 in the image below.)



[68-point facial landmark]

[Results of Eye detection]
(Eyes detected are marked as blue)



[Table 2.] Face-Detector + 68-point Eye Detector Comparison (FPS)

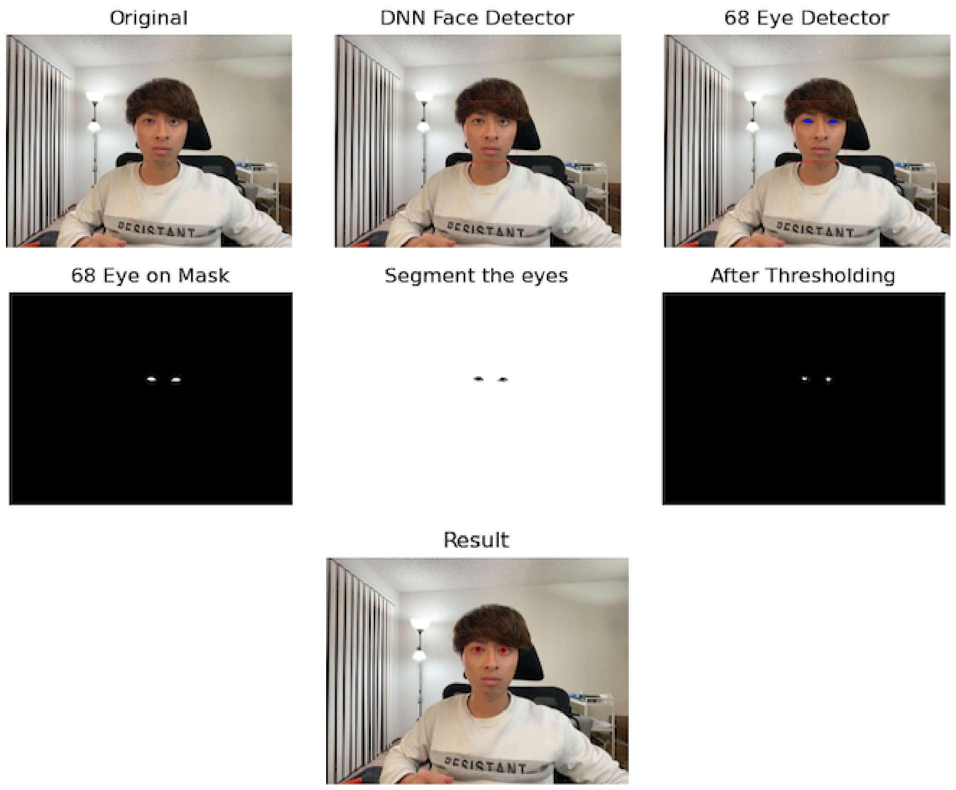
Method	HAAR Cascade	HOG	MTCNN	DNN
FPS	1.84	3.62	0.61	9.91

Pupil Localization

After getting the coordinates of the eyes, the final step would be to point out the pupils of two eyes, which would be the centroid of the eyes. Below, I referred to [4] to implement the step-by-step method in order to get the pupils out of the eyeballs.

1. Based on the left and right eye coordinates, filled the eye areas using cv2.fillConvexPoly, which would connect the 6 points of one eye, and fill the color inside the line.
2. In order to cover a little bit more area of the eyes for detection, use dilation provided by cv2.dilate to segment out the eye areas from the original image.
3. Threshold the two eyeballs out of the rest of the eye by adjusting the thresholding value.
4. By finding the largest contours on both sides of the face using cv2.contourArea are the eyeballs.
5. The centers of the eyeballs, which are pupils, would then be the centroid got by cv2.moments.

[Results of Pupil localization]

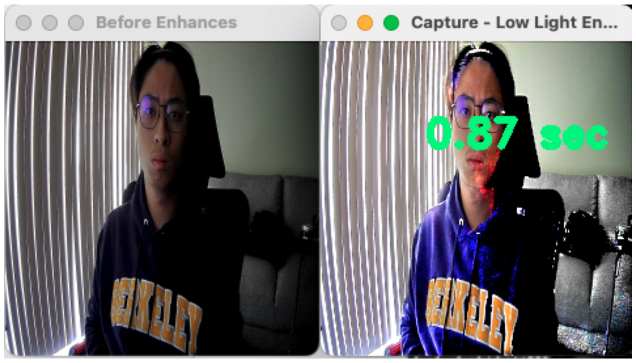


Low Light Enhancement

In this semester, we learned two ways to enhance low-light images, which are Enlighten GAN and Zero Deep Curve Estimation. On top of that, I found two conventional methods, LIME and Dual Illumination Estimation, to serve as the low light enhancement method within my eye-tracking system. Because of the lack of GPU or training resources and one of the goals of the eye-tracking system is to run on mobile devices, all of the methods would be implemented on the CPU. Below, I implemented three low light enhancement methods: LIME, DUAL Illumination Estimation, and Zero Deep Curve Estimation.

1. LIME: Low-Light Image Enhancement via Illumination Map Estimation

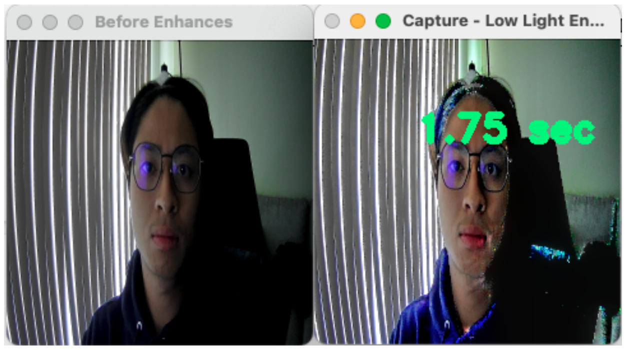
In LIME, the illumination of each pixel is estimated individually through finding the maximum value in RGB channels first, and the initial illumination would then be refined by imposing a structure prior upon it as the final illumination. Last, the enhancement would be achieved by using the well-constructed illumination map.



[Result of LIME (Image size = 320x240x3)]

2. <u>Dual Illumination Estimation</u>

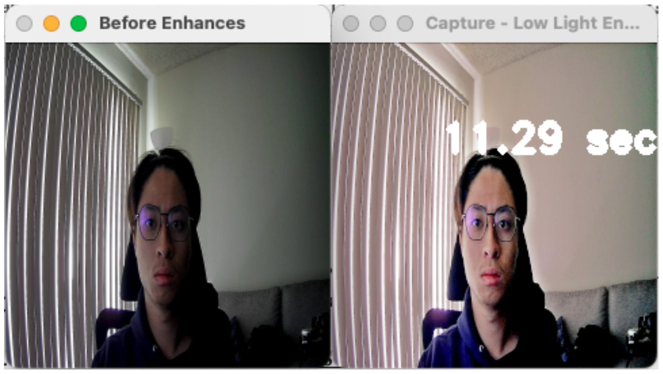
The method mainly deal with the images under various exposure conditions. Dual illumination estimation stands for under and over-exposure correction as the illumination estimation of the input image and the inverted input image. By performing dual illumination estimation, two intermediate exposure correction results for the input image would be obtained, with one fixes the underexposed regions and the other one restores the overexposed regions. The last step is to adaptively blend the visually best exposed parts in the two intermediate exposure correction images and the input image into a globally well-exposed image.



[Result of Dual Illumination Est (Image size = 320x240x3)]

3. <u>Zero Deep Curve Estimation</u>

ZeroDCE model is trained by four loss functions, including Spatial Consistency Loss, Exposure Control Loss, Color Constancy Loss, and Illumination Smoothness Loss, on a lightweight deep network to estimate pixel-wise and high-order well-designed curves for dynamic range adjustment for a given input image.



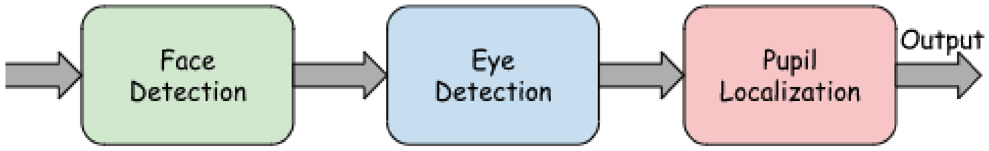
[Result of Zero DCE (Image size = 320x240x3)]

[Table 3.] Low Light Enhancement Comparison (sec/frame)

Method	LIME	Dual Illum. Est.	Zero DCE
Sec per frame	0.87	1.75	11.29

Eye-Tracking System Under the Normal Enviroment

First, I implemented the eye-tracking system without the low light enhancement module to see how fast could the face detector and eye detector reach. The size of the input caught by the webcam is **1080x1920x3**.

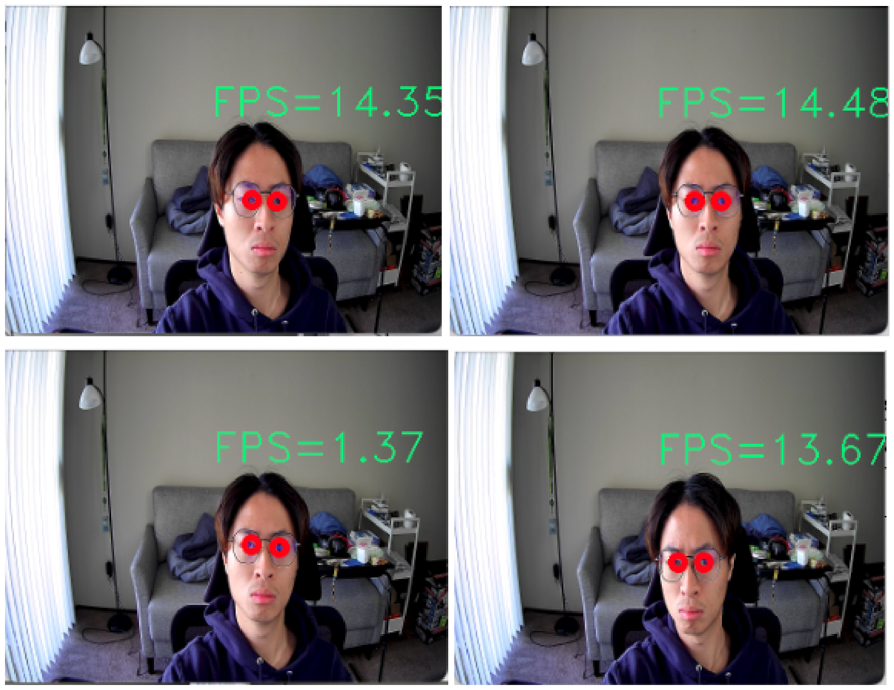


[Result of Eyetracking System: (ul) Haar, (ur) HOG, (bl) MTCNN, (br) DNN]

[Table 4.] Face-Detector + 68-point Eye Detector + Pupil Detection Comparison (FPS)

Method	HAAR Cascade	HOG	MTCNN	DNN
FPS	1.39	2.18	0.52	3.34

From the result above, none of them could be considered as the real time (>20fps); hence, I tried to down size the input image caught by the webcam to **320x240x3**, and the results are shown below.



[Result of Eyetracking System: (ul) Haar, (ur) HOG, (bl) MTCNN, (br) DNN]

[Table 5.] Face-Detector + 68-point Eye Detector + Pupil Detection Comparison (FPS)

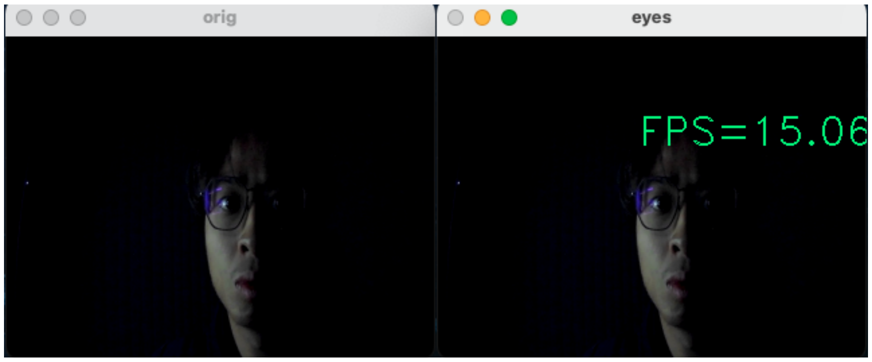
Method	HAAR Cascade	HOG	MTCNN	DNN
FPS	14.35	14.48	1.37	13.67

Even though the fps still under 20 fps, HAAR, HOG, and DNN face detector with 68-point facial landmark eye detector could reach above 14 fps, which is acceptable to be consider as eye-tracking system. On the other hand, because MTCNN runs 10x slower than others, it would not be considered to serve as face detector within the eye-tracking system.

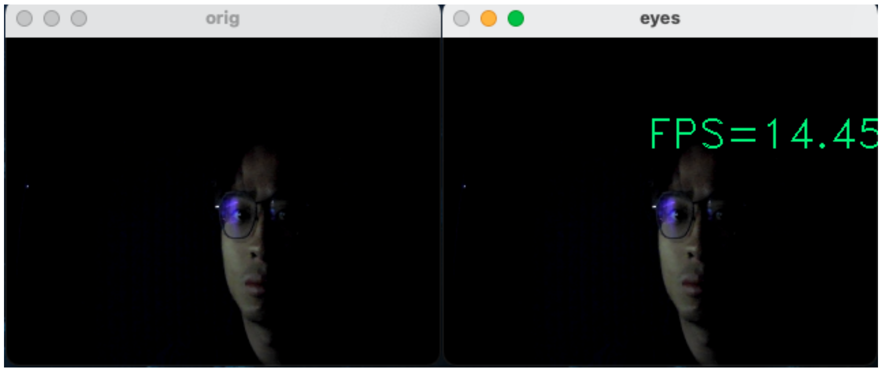
Eye-Tracking System Under the Dim Light Enviroment

Next, let's test the system under the dim light environment. First, I would do the experiments on three different face detector, HAAR, HOG, and DNN, to see which detector could perform the best under the low light environment. Based on the result, I would then combine the low light enhancement module within the eye-tracking system.

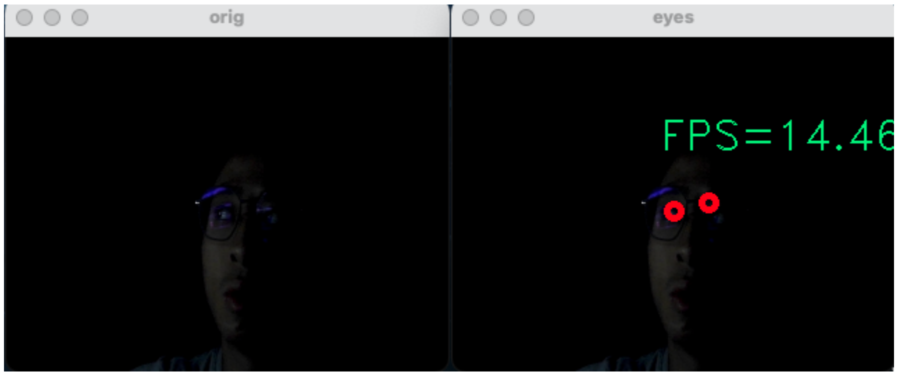
[Result of HAAR face detector + 68-facial landmark eye detector]



[Result of HOG face detector + 68-facial landmark eye detector]



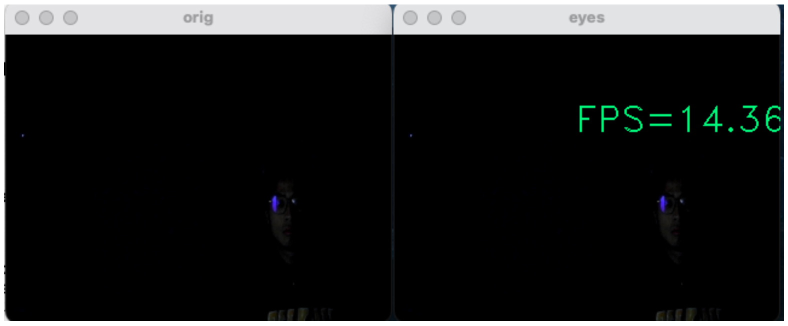
[Result of DNN face detector + 68-facial landmark eye detector]



It is clear that only DNN face detector, could detect the face and eyes under the extreme low light environment, and with high frame rate. Hence, the next step I would try the DNN eye-tracking system under even worse lighting conditions, and add the low light enhancement module to improve the result.

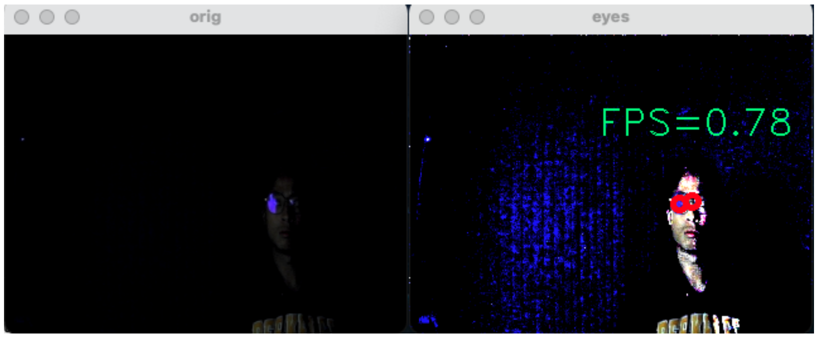
The experiment shown below, the DNN eye-tracking system could find any face within the extremely low light environment, let alone the eyes.

[Result of DNN face detector + 68-facial landmark eye detector]

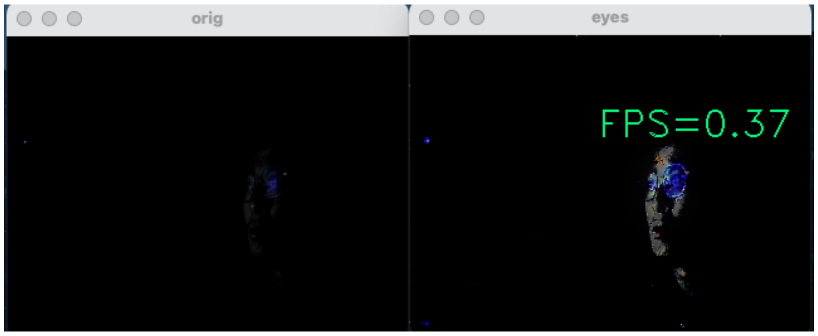


Based on the **Table 3.**, I would choose LIME and DUAL to serve as my low light enhancement models because they run much faster then ZeroDCE on CPU even though ZeroDCE could get the higher quality of the enhanced image. Below, I did the experiments on LIME and DUAL model under the same extreme conditions.

[Result of LIME low light enhancement + DNN face detector + 68-facial landmark eye detector]



[Result of DUAL low light enhancement + DNN face detector + 68-facial landmark eye detector]



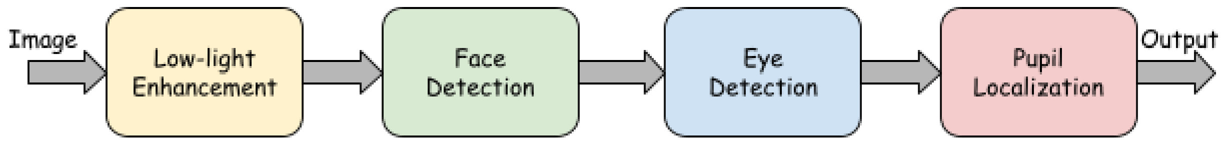
As the result, LIME low light enhancement module would be chosen as the first stage of my eye-tracking system because it could reach higher fps as well as better detection result.

Conclusion

In this project, I aimed to implement a real-time eye tracking system which could perform well under both normal and dim light environment.

First, I experimented with four different face detector (HAAR, HOG, MTCNN, DNN) with the 68-point facial landmark eye detector with the pupil localization algorithm follows. Next, I eliminated MTCNN face detector because of the slow processing time, and chose DNN as the face detector based on the experiment under the dim environment. Last, I performed the experiments on LIME and DUAL low light enhancement models and picked LIME as my enhancement model.

All in all, my low light enhancement eye-tracking system serves as below.



Stage	Low-light Enhancement	Face Detection	Eye Detection	Pupil Localization
Module	LIME	DNN	68-point facial landmark	Contour+Moments

[Low-light Enhancement Eye-tracking System]

Reference

[1] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.
[2] K. Zhang, Z. Zhang, Z. Li, Y. Qiao, “Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks,” arXiv:1604.02878
[3] Face Detection Models: Which to Use and Why?
[4] Real-time eye tracking using OpenCV and Dlib
[5] Q. Zhang, Y. Nie, W. Zheng, “Dual Illumination Estimation for Robust Exposure Correction,” arXiv:1910.13688
[6] X. Guo, Y. Li and H. Ling, “LIME: Low-Light Image Enhancement via Illumination Map Estimation,” in IEEE Transactions on Image Processing, vol. 26, no. 2, pp. 982-993, Feb. 2017, doi: 10.1109/TIP.2016.2639450.
[7] Low Light Image Enhancement Python Implementation
[8] C. Guo, C. Li, J. Guo, C. Change Loy, J. Hou, S. Kwong, R. Cong, “Zero-Reference Deep Curve Estimation for Low-Light Image Enhancement,” arXiv:2001.06826
[9] Y. Jiang, X. Gong, D. Liu, Y. Cheng, C. Fang, X. Shen, J. Yang, P. Zhou, Z. Wang, “EnlightenGAN: Deep Light Enhancement Without Paired Supervision,” arXiv:1906.06972
[10] Barsky Lab
[11] CNN based face detector from dlib
[12] DNN-based Face Detection And Recognition
[13] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1867-1874, doi: 10.1109/CVPR.2014.241.