**Iris Tracking and Data Recording**

Robert Taylor and Jesse Claiborne III

Sambit Bhattacharya, Mingxian Jin

CSC 490 Fall Semester 2022

Abstract

Eye tracking has been around for some time now, but the software behind these devices have constantly been improving with different companies creating different versions that are streamlined and optimized.Our purpose for this project is to develop eye tracking programs that can test for mental and neurological disorders. Unfortunately, we aren’t a big company that can dump hundreds of dollars into creating these beasts of machines. Instead, we took the more amateur approach with cheap external webcams and some python code. Our objective was simple yet complicated. Take the movements of the iris, the white space around it, and the overall eye, and track the movement of each component, recording the data, and saving it for future machine learning/AI use. In order to achieve this goal, we needed to understand what kind of software and hardware could be used to do so. At first, we were advised to look at PyGaze, a free open use software that could handle the task. Unfortunately after some research, we found that it needed to be used with specific eye trackers. So we looked up the next best thing. OpenCV (computer vision) is a python library dedicated to gathering facial map data and training AI to be able to eye track.

**Project: Start**

Our goal was to develop an eye tracker and create a calibration test to be able to record the data for future AI training use. So we started looking. Initially we were pointed in the direction of [PyGaze](http://www.pygaze.org/documentation/), a free, open use python library that could possibly give us traction for this project.

Unfortunately, after some research and wasted time, we found that the library was actually a tool kit designed around specific eye tracking hardware and software. The creator had even said it himself that, quote, “PyGaze can't do webcam eye-tracking. You don't "just" track an eye using your webcam”. We were hitting the halfway point and needed an alternative fast. That’s when we came across OpenCV.

**Required Software**

In order for us to get this stuff up and running, we needed specific python libraries and software for everything to work properly. First, we needed the OpenCV library which could be downloaded straight from their website and loaded to the computer. We then installed the following software and libraries:

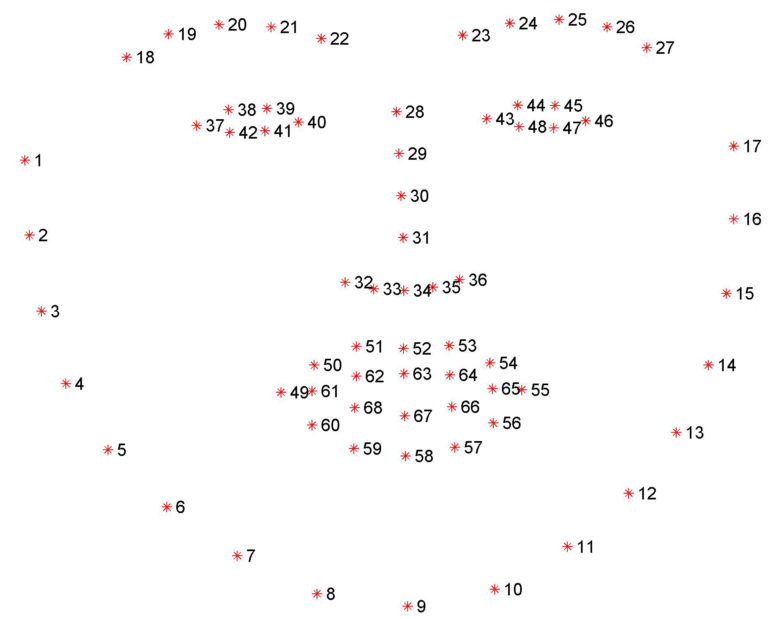
* Python 3.10
* Standard python libraries
* Dlib
* Mediapipe
* Numpy
* CSV
* Pandas (just in case)
* Pillow
* shape\_predictor\_68\_face\_landmarks.dat
* PsychoPY
* CMake
* Preferred Compiler (VS, VS code, Python IDE, Anaconda, etc.)

**Progress**

OpenCV is a library hosting thousands of computer vision algorithms. Anything from object detection, to image processing and analysis could be done. So at this point we began to split up our tasks. Robert would focus on the test potion, while I created the eye tracking algorithm.

I got to doing research on if this was possible or not. After some googling and youtube searching, I found a couple of great resources that I could use. To start, I found a youtuber that introduced me to the concept of object detection in OpenCV using haar cascades. Haar cascades are essentially data files created in order to detect objects. In this video, I was introduced to the functions of the OpenCV library and how they worked. I learned how to capture a video or image source, and then from there, create barriers around certain portions of the objects using the haar cascades. Example being a live webcam feed of my face and drawing boxes around my head and eyes.

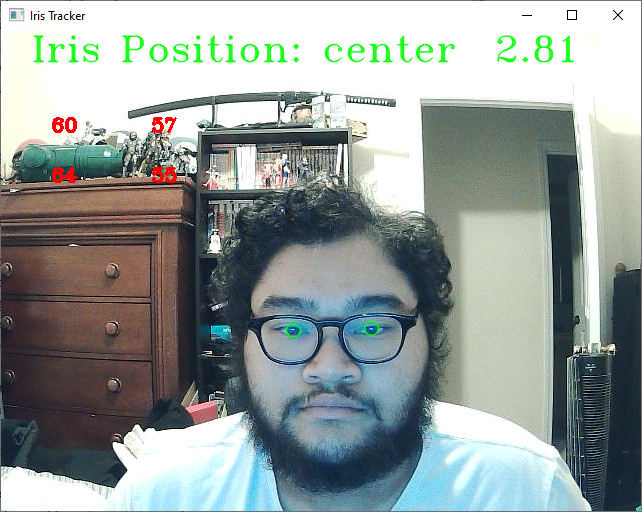
On the right track, I needed to refine this program and make it more focused on the eyes itself. That’s when I found another youtube video on using the mediapipe library to get a face mesh template (seen below) and read the landmarks of the eyes.



Thanks to that same video, source code was provided for me to start my build. In the code, the author had already calculated the halfway points of the eyes based off of the corner landmarks of the eye, and then drew a circle frame around the iris and displayed

the current position whether it was left, right, or center (images below and right).

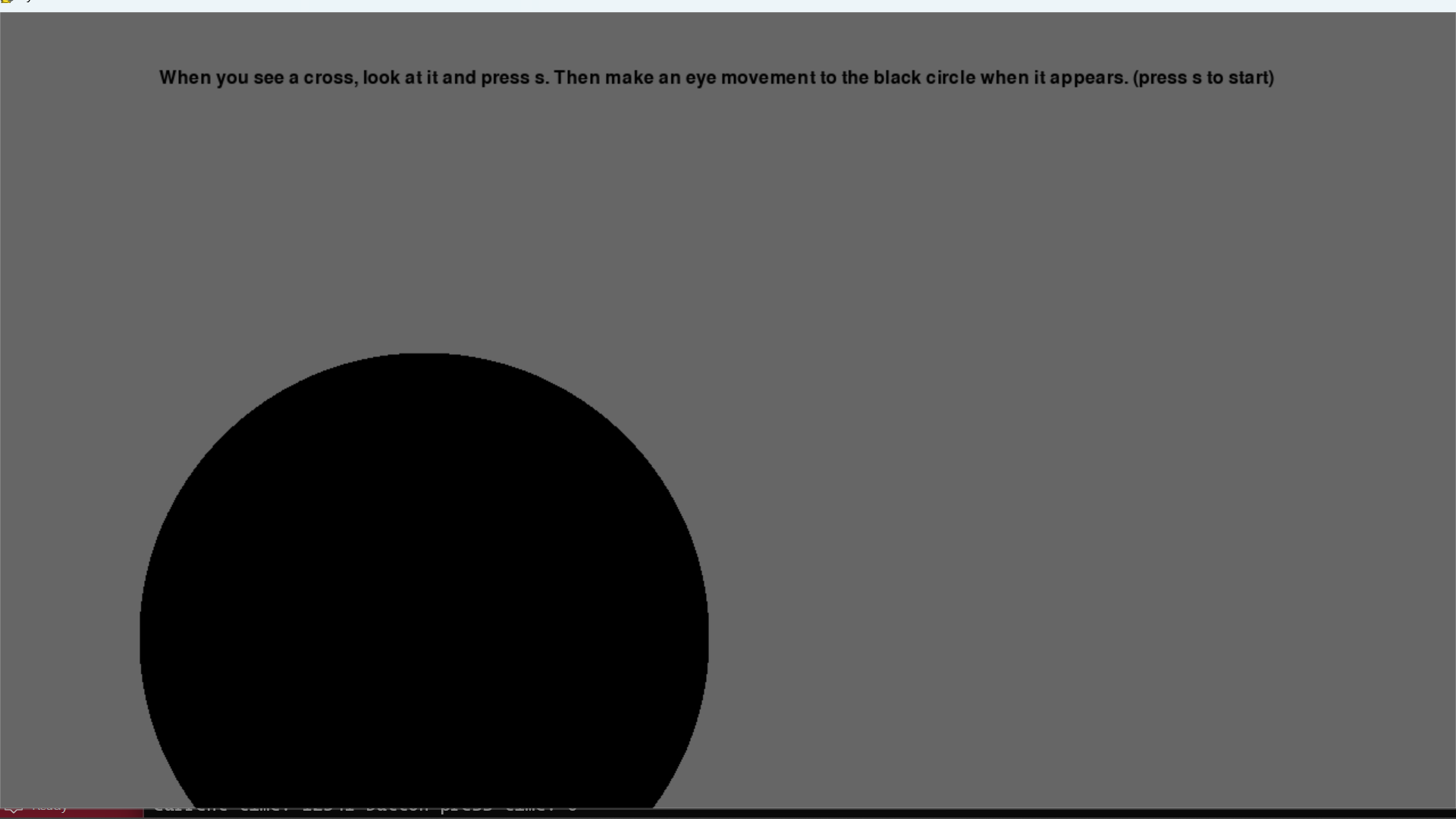
 



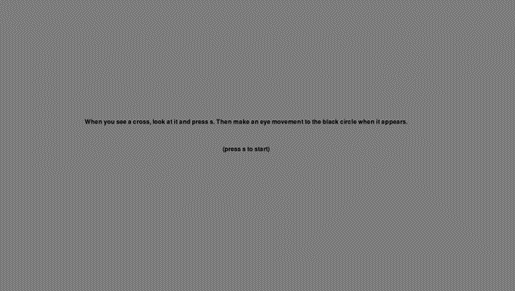
For Robert’s portion of the project, he focused on a test for the eye tracking portion. When he started he used python to even see if it was possible. The end result was a very simple test with a circle that could bounce around from one corner of the screen to another. From there he attempted to make the test with the pygame libraries at the recommendation from our advisor.

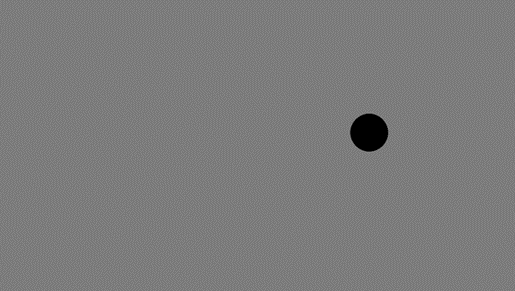
Which came with a heap of problems which made it that much harder to code.

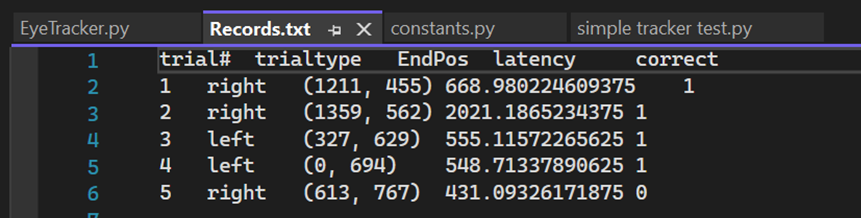
Pygame is a set of libraries and functions that are specifically used to design video games. This language made sense because the test is very much like a game. Originally it was promising, finding tutorials on making the base framework for the system was very easy to find. Problem 1 comes in where you cannot make black dots within pygame you have to import any asset that would need to be used. Secondly from what Robert had researched it is hard to make separate windows so for the sake of time we decided to move away from this model. Results of this version are shown below.



From this is the point where we realized that pygaze was not used to create eye trackers but it can be used to create eye tracker tests. By cross referencing with another test made with pygaze Robert made a test with the given specifications. While Robert followed the example closely he made tweaks along the way to help make the test more user friendly and easy to edit. Turn off the mouse so it doesn’t display on screen and a way to edit who many trials you do are just a few of the edits Robert made. The last challenge that he had was getting it to interface with the eye tracker but across multiple tests we could not get it working but in the end the test was made to the specifications of the proposal that was given by the advisor.

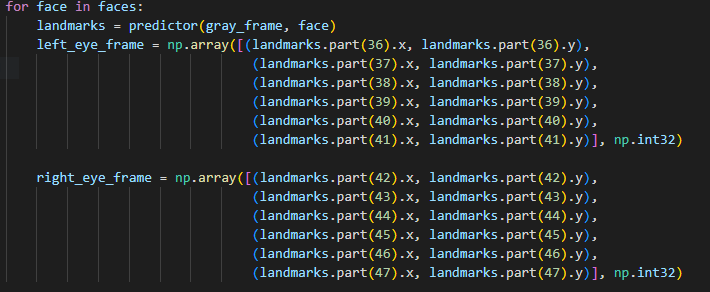
****

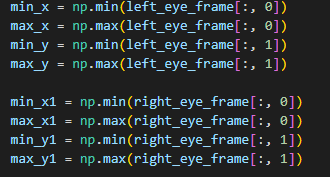
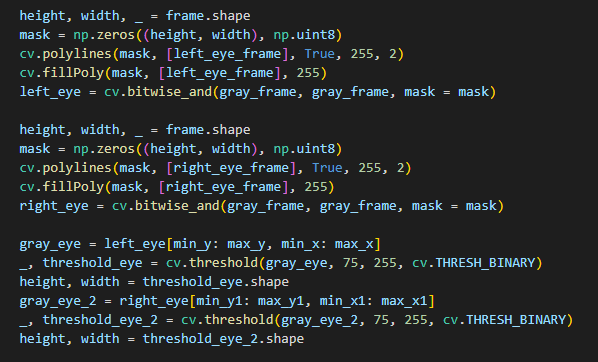




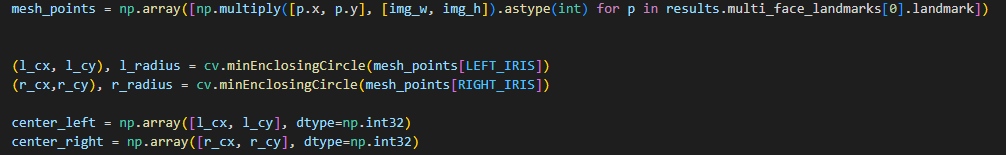
**Algorithms**

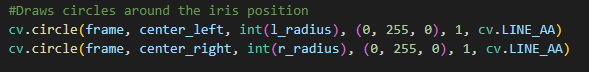
In order to get the eyes registered in a separate frame, I needed to make an array of the facial landmarks provided in the dlib facial landmark data file. Using the numpy library made this possible for the code to read two arrays containing facial landmarks for both the left and right eye and grab that area specific to each, by calculating the minimum and maximum X and Y coordinates of the eye and create that area in

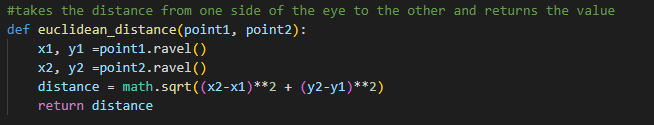
two separate cv.imshow( ) frames (shown here).

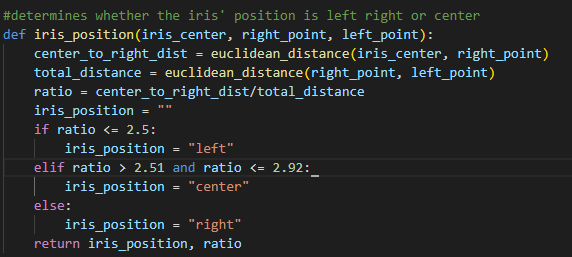
The frame for each individual eye is created by making a gray threshold mask of the rgb frame (the current feed from the source, like a webcam, video, or image) and then calculate the height and width of the frame by taking the min and max X and Y values of each eye. This then creates a smaller, zoomed in frame of the eye and pupil, and applies a gray filter to better read the contours and the white space of the eye (shown top right).  To get the location of the pupil, we use the math library to calculate the midpoint of the eye based on the farthest left and right points of each eye itself. After calculating the midpoint, the next objective was to draw the circle around the iris and determine the gaze position. To do this, we take the array from the mediapipe portion of the landmarks and create an enclosing circle based on the left and right iris points and then took those midpoint values and turn them into a ratio. If the ratio hits a certain value, the position displays either left, right, or center(seen below).

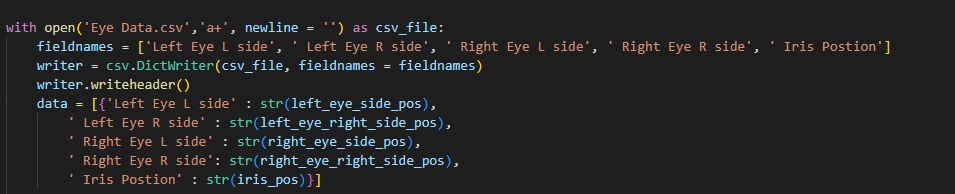


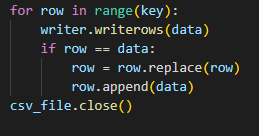








Finally, we seeked to put all of the data into a comma separated value file. In order to do so, I imported the csv library to open a file called Eye Data.csv. I then wrote the field names and set the list as a dictionary. I took the keys and set the values to strings just in case to and then wrote a for loop to try and iterate each row based off the kill key for the program. In an attempt to make sure that each line is different and constantly updating, I would have the loop read the data, see if the line was the same, and if it was, replace the row and append the new data, This however did not work like I wanted it to (shown below).



**Challenges and Future Improvements**

Many of the challenges we faced were having to learn how to use OpenCV and getting the tracker and test to work together. There were also issues with outputting the data to a CSV file. The main issue with having the two programs work with each other was trying to get the eye tracker to run from the constants.py file, but whenever this happened, the program overall would break. Another solution that I thought may work was combining both into one large program and executing that way. The issue with this however, was that the programs would run individually and the test would still not use the tracker. There’s no doubt that both of these programs need to be optimized for future use. Improvements I’d like to note are trying to get the amount of code being used shortened down and run faster on any system. Fixing the data output to a CSV file and have it record the numbers real time. Streamlining the test to incorporate profiles for users and saving that data as well.

**Conclusion**

Overall, this project was complicated, yet moderately understandable. We learned a good deal about how computer vision, artificial intelligence, and machine learning co-exist to make this sort of software and technology. The application of the research done was time consuming, but overall, we felt great knowing that we had a product that works, for being amateurs. A special thank you to our advisor, Sambit Bhattacharya, for giving us the idea for this project. He gave us a good deal of information to start out with and helped put us on the right track for the end goal. The following page will contain links to every video and site used for researching this topic, as well as a link to the github repository that users can view and download this project from.

**References, Research, Source material, Videos, etc.**

Forum where PyGaze creator responded to user issue about “inability” to use webcam for eye tracking with PyGaze:

* <https://forum.cogsci.nl/discussion/1512/solved-pygaze-tutorial-not-working-forgot-to-turn-off-dummymode>

Video References for code material:

* <https://www.youtube.com/watch?v=DNKAvDeqH_Y>
* <https://www.youtube.com/watch?v=8CIxfcbGU3s>
* <https://www.youtube.com/watch?v=kbdbZFT9NQI> (contour functionality)
* <https://www.youtube.com/watch?v=qpFrg4gN4Mg&t=3294s> (referenced but did not use)
* <https://www.youtube.com/watch?v=88HdqNDQsEk&t=239s> (introduced OpenCV object detection)
* <https://www.youtube.com/watch?v=Jvf5y21ZqtQ&t=348s> (threshold window)
* <https://www.youtube.com/watch?v=obKG1SXp76Y>
* <https://www.youtube.com/watch?v=zDN-wwd5cfo> (referenced but did not use)

Source code referenced and/or used:

* <https://github.com/Asadullah-Dal17/Iris-python/blob/master/Mediapip-Eyes-tracking-master%2030.05.2022/eyes_tracking/rastreamento-iris.py> (used this source code and currently building off of it)
* <https://github.com/Asadullah-Dal17/Eyes-Position-Estimator-Mediapipe>
* <https://github.com/esdalmaijer/webcam-eyetracker>
* <https://github.com/stepacool/Eye-Tracker>

Other resources/references:

* <https://pysource.com/2019/01/04/eye-motion-tracking-opencv-with-python/>
* <https://google.github.io/mediapipe/>
* <https://google.github.io/mediapipe/solutions/face_mesh>
* <https://google.github.io/mediapipe/solutions/iris>

Software and libraries:

* <https://pypi.org/project/cmake/>
* <https://opencv.org>
* <https://www.psychopy.org/download.html>
* <https://github.com/coneypo/Dlib_install>
* <https://google.github.io/mediapipe/>
* <https://pypi.org/project/python-csv/>
* <https://numpy.org/install/>
* <https://pypi.org/project/Pillow/>
* <https://pypi.org/project/pandas/>