Networked Eye Tracking Using Commodity Hardware

Vinh Nguyen (WIM), Jananan Mithrakumar (4-units)
vnguyen5@stanford.edu, janamith@stanford.edu

Abstract

Gaze tracking and other forms of eye tracking have many useful applications in VR and accessibility, but have not been made commonplace despite a fairly long history of research. Current commercial solutions for gaze tracking are few and expensive, and replication of their techniques is difficult and potentially invasive. In this paper we build a low-cost, non-invasive eye tracking system which can be made from commodity hardware.

1. Introduction

Gaze tracking or gaze estimation, is a subset of eye tracking focused on pinpointing where the eyes pointed on a 2D screen. Research in eye tracking goes as far back as the 19th century, where it was conducted to determine where and for how long readers fixed their eyes on different parts of a text. In virtual reality, gaze tracking has many applications, which include foveated rendering, dynamic focus adjustment. These are integral for creating immersive and photo-realistic virtual reality, but the benefits of gaze tracking extend beyond this. Gaze tracking data can be analyzed to understand a user's reaction to virtual stimuli and improve the virtual experience or answer research questions; in addition, it can be primary method of user input for disability-mitigating applications.

Current commercial gaze tracking solutions for virtual reality (i.e Tobii, Pupil Labs) are few and often very expensive, which impedes its integration into consumer hardware. There are several established methods for gaze tracking for desktop or mobile applications, even including some which use commonplace devices such as webcams and phone cameras; however, none of these solutions have been successfully and cost-effectively integrated into VR. For gaze tracking integration to be successful, it must be accurate, non-invasive, and have low latency. Accuracy and latency are paramount for minimizing discomfort; one work estimates the error threshold for discomfort in VR to be $1.4°$. [2]. In addition, existing methods for gaze tracking are or may be invasive when integrated into VR; this work addresses the above challenges through a safe, affordable DIY solution for gaze tracking.

2. Related Work

There are several approaches to eye tracking in the literature, which can be divided based on the input data collected: Optical tracking, electro-ocular tracking, and contact tracking. Contact tracking involves a special contact lens being inserted into the eye. Magnetic fields induce currents in a coil embedded in the lens differently based on the orientation of the eye, which can be measured. This method has the best accuracy of the three methods, and involves the least computation.[5][10]. However, this method so invasive that users typically cannot comfortably wear the lens for longer than 30 minutes without anesthesia, so this is not a viable method for consumer VR.

2.1. Electro-ocular Tracking

This method involves placing electrodes around the eye at key points relative to the muscles which move the eye, and using measurements to determine where the eye is gazing. It is able to track horizontal movements within reasonable accuracy (within 1-2 degrees), but vertical movements are inaccurate and are sensitive to illumination [4] [1][10]. If improved, this could provide a less-computationally intensive means of performing gaze tracking, but in its current state is not viable for our purposes.

2.2. Optical Tracking

Optical tracking involves one or more cameras pointed at the eye and software to extract the orientation of the eye. The A common approach is done using a near-infrared camera and an infrared light source pointed at the eye; reflections off the cornea can be detected by the camera and used to determine pupil location and then estimate gaze. This approach is inexpensive and has become the most popular due to advancements in processor speed and power within the last two decades[1][3]. This technique has been applied to head mounted displays to the accuracy of $.95°$ in a research setting. [7]. The existing commercial trackers use some variation of this; however, the light sources are extensively characterized for eye safety. Because the safety of IR LEDs

at such a close distance to eye is not easily verified, active IR is not viable for a DIY gaze tracking system. Therefore, for our system we will only use RGB images and rely on passive lighting.

While there are several examples in the literature of eye tracking with RGB web camera imaging[9] [6], many of these only demonstrate pupil tracking or eye localization. One demonstrated gaze tracking: Krafka et al's 'iTracker' used a convolutional neural network trained on over 1.5 million images to perform gaze estimation from camera images captured by iPhones and iPads. They reported an error of 1.04cm on phones and 1.69cm on tablets. Assuming a viewing distance of 30cm, these correspond to errors of $1.98°$ and $3.22°$. The authors also report that real-time inference was estimated to run at 10-15fps at best on mobile devices, which demonstrates the challenge of creating a system that can run smoothly in real time.

There are no examples in the literature of this technique being applied to commercial HMDs using RGB imaging. A system that can robustly perform eye tracking using only RGB imaging could accelerate the integration of eye tracking into commercial VR systems.

3. Our System

For this project, we focused on performing gaze-tracking using a single eye to reduce the amount of hardware that we had to purchase. With the hardware choices we made, extending the system to make use of two eyes would have only involved increasing the number of cameras used.

3.1. Hardware

We decided that our platform of choice for implementing a low-cost, low latency non-invasive gaze tracking system was the Raspberry Pi 3 Model B+. We chose this platform for the following reasons:

1. Raspberry Pis are ubiquitous in DIY projects and are very easy to use

2. They are relatively low cost ($35)

3. There are existing camera add-ons which interface very easily with the Pi

4. The Pi provides more than enough compute with a 1.4 GHz quad core processor

5. They are lightweight, portable and have wireless connectivity capabilities

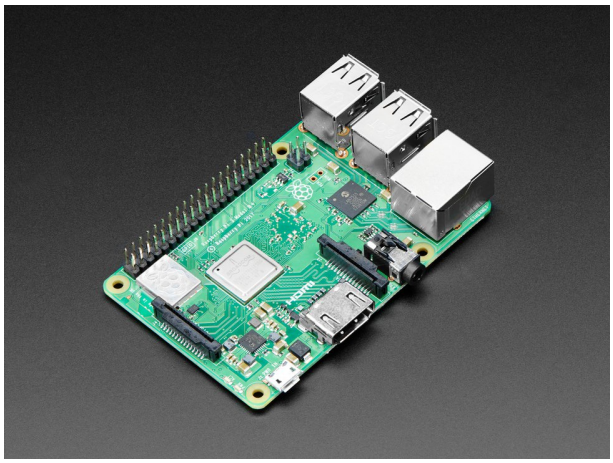


Figure 1. Raspberry Pi

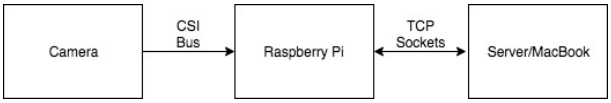
For our camera we opted for a Raspberry Pi "Spy Camera" product sold on Adafruit. This camera was great for our application due to its long cable and small housing size. It also provided more than enough resolution with a native resolution of 5 Megapixels. The camera interfaced with the Pi via a high bandwidth CSI bus.

Figure 2. Raspberry Pi Spy Camera

To facilitate eye-tracking we mounted our camera onto a glass frame, making sure that the images captured were straight-on images of the eye and the amount of view occlusion was minimized. All graphical rendering was carried out on a MacBook Pro using WebGL.

3.2. System Integration and Networking

Our system involved the following communication flow:



Communication between the Raspberry Pi and the MacBook was facilitated through Ethernet and TCP network sockets via the python socket abstraction. This allowed two-way communication between the Pi and the host computer. We opted to go wired to maintain lower latency and due to its relative simplicity. However, further work would be adapting the system to use the on-board wireless capabilities of the Pi.

Figure 3. Full Gaze tracking hardware setup

4. Methodology

4.1. Hough Transform for Eye Center Detection

We initially attempted to perform eye center detection. This was done by using a Haar cascades classifier to first localize the bounding box of the eye. This region was then cropped from the frame and a Gaussian blur was applied. A Hough circle transform was then used to detect the iris. Once a circle was fitted to the iris, the pupil location was approximated as the center of the circle. These algorithms were all readily available in OpenCV.

4.1.1 Parameter Tuning

The accuracy of the eye-center detection greatly depended on the parameters supplied to the Hough circle transform. Namely, the threshold to the canny edge detector, the accumulator threshold, min circle radius and max circle radius had the most significant effect on the eye center detection accuracy.

To produce optimal eye center tracking, these parameters had to be tuned for different camera to eye distances, camera angle and illumination. The parameters also had to be tuned for different users' eyes. Therefore, the algorithm was quite cumbersome to use

4.1.2 Results

Our eye center detection algorithm yielded relatively good results for small shifts in pupil position from the center. However, accuracy dropped considerably for larger shifts where the iris started to become occluded or started to become more elliptical in the camera view. Furthermore, many constraints had to be applied. Firstly, the Hough circle parameters had to be carefully tuned for the different image features discussed above. Secondly, the eye-center detector only worked over a limited range of eye movement. Thirdly, the detector also led to a considerable drop in the frame-rate of the camera

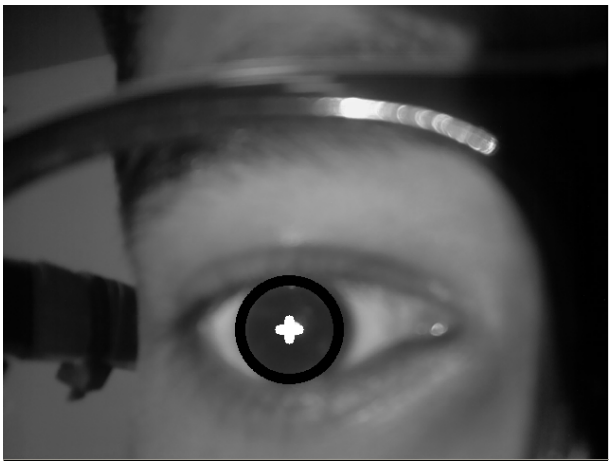


Figure 4. Good detect

Figure 5. Good detect

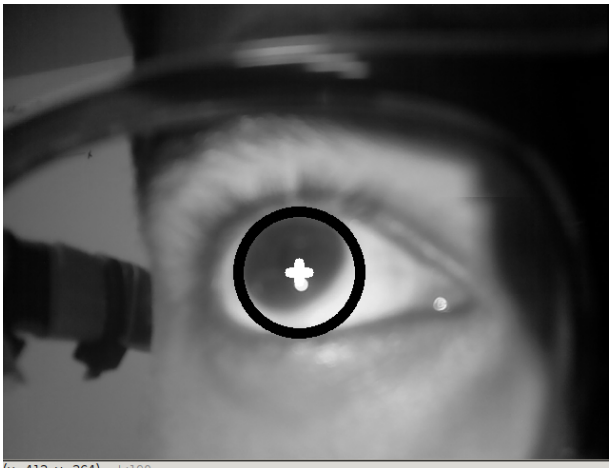


Figure 6. Bad detect

As we can see in figure 6, occlusion of the iris by the eye-lids and the warping of the iris into an elliptical shape due to the curvature of the eyeball causes the hough circle transform to become quite inaccurate

Due to the various constraints that must be imposed on the image capture process, the sensitivity to hyper parameters, the inaccuracy of the method to large off-center shifts in pupil position and the drop in frame-rate, we decided not to use this method to implement gaze tracking

4.2. K-Nearest-Neighbors for Gaze tracking

Our gaze tracking system runs directly on the Raspberry Pi, which captures eye images and outputs the estimated (x,y) gaze on the screen to a network socket, where it can be accessed by the system driving the application.

For our gaze estimator, we decided not to use a CNN as is the method of choice in the literature. This is mainly due to the lack of easily-applicable frameworks and the time constraints of the project. The eye tracking datasets we found in the literature contained images captured at much larger distances than those produced by our system; this made it unlikely that a system trained on one of these datasets would generalize to our data. Models which take advantage of near IR illumination artifacts (namely, the corneal reflection) were also not applicable to our system.

Due to these constraints, we decided to build our own model with a k-nearest neighbor (KNN) estimator with principle components analysis (PCA) for feature extraction. We generated a training dataset of calibration images compiled before the eye-tracking application is run. This follows an approach similar to those of several publications on gaze estimation. [8][11]. The calibration process is described in more detail in the Calibration subsection.

The training set images preprocessed using openCV and numpy in Python3. They are first converted to grayscale and then Haar Cascade classifiers are used to crop the image into a fixed size containing only the eye. We then use Principle Components Analysis to compress the images, producing the final training dataset. The principle components are saved. ¡include: example images of eye, and preprocessed versions¿

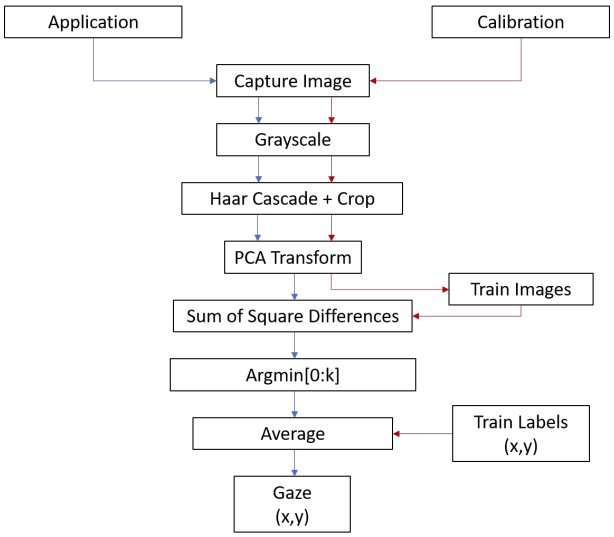


Figure 7. Diagram of gaze estimation algorithm.

Once the training dataset is compiled, the real-time eye tracker captures images of the eye, compressing the image based on the PCA weights of the training set, and finds the sum of square differences between the compressed real-time (test) image and each of the train images. The k images which have the smallest distance from the test image are then selected and the corresponding gaze locations of these images are averaged to produce the prediction of user

gaze. Figure 4 shows a flow diagram of the gaze predictor.

4.2.1 Model Tuning and Optimization

There were several parameters to the gaze tracking model that we considered for optimization. These can be divided into the following classes:

1. Training Set Parameters.

   Due to the limited RAM size of 1GB, the training set had size constraints which were parameterized by the number of points for calibration, the number of training images to take at each point, the size of the image crop, the number of principle components for image compression, and whether columnwise or pixelwise PCA was performed.

2. Model Parameters

   These include the number of neighbors k, and whether to perform a weighted average over the distances of the nearest neighbors. Indirectly related were the parameters for the Haar Cascade classifiers and the distance of the camera from the eye.

Using these parameters, we define the predictor architecture using the following:

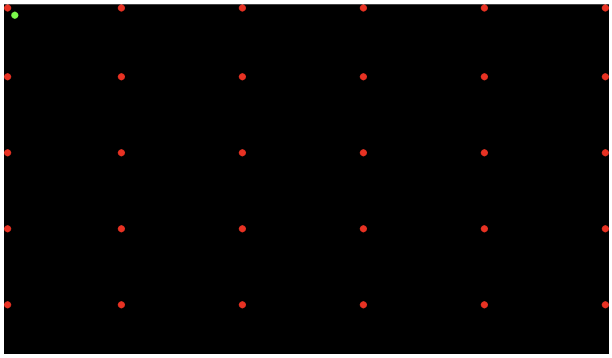
$$\textbf{Pred\_KNN\_params} = [N_{img}, N_{cal}, l_{crop}, w_{crop}, N_{PCA}, PCA, k, \mu, \lambda_{H,1}, \lambda_{H,2}]$$

where N_cal is the number of calibration points, P is 1 if pixelwise PCA is used, 0 if columnwise PCA is used, and -1 if no PCA is performed; μ is 1 if weighted average is used and 0 if the simple mean is used. $\lambda_{H,1}, \lambda_{H,2}$ refer to the parameters of openCV's Haar Cascade classifier. For our analysis we use default parameters for the Haar cascade to simplify the search space.

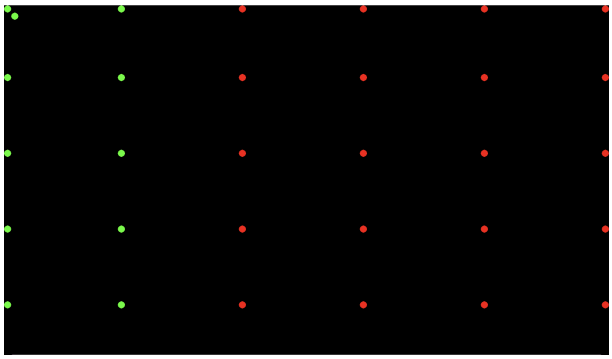
4.2.2 Calibration

To capture the training images for our k-nearest neighbor algorithm we use the following calibration procedure. All rendering in this calibration step was done using pygame, an open-source set of python modules designed for writing games.

1. Render a uniform grid of red points on the Mac-Book screen and an off-grid green point



2. The user must focus there eyes on each point and press the point until it turns green



3. A fixed number of snapshots of the user's eye will then be taken at each grid point

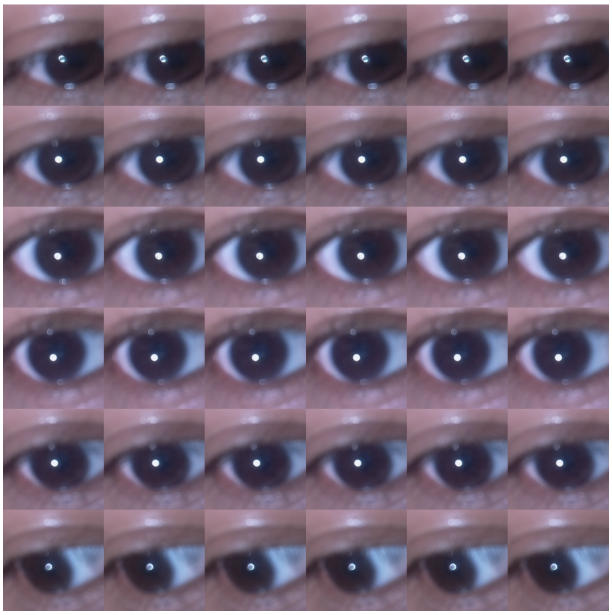


Figure 8. A selection of training data images.

4. This is repeated for all the remaining red points on the screen

5. The user will then click the off-grid green point to finish the calibration

Once calibration is complete, we will have acquired a training set of eye images which we use in our k-nearest neighbors algorithm

4.3. Model Evaluation

The accuracy of the model is measured in terms of distance between the true point of focus and the predicted gaze location. Assuming a fixed viewing distance, this can be converted to an error in visual angle using the formula

$$\epsilon = atan(\frac{\sqrt{(\hat{x}-x)^2+(\hat{y}-y)^2}\cdot d_p}{D})$$

where $(\hat{x}, \hat{y})$ is the predicted gaze location in pixels, d_p is the pixel pitch in meters, and D is the viewing distance in meters. For our evaluation, we fix D to be 50cm. Our evaluation collects 108 images at 36 randomly generated fixation points and records the predicted gaze location with the true fixation point coordinates to compute the error. The error is analyzed across different model configurations and minimized to arrive at the best model.

Other important metrics for evaluating the model include throughput (how many predictions per second can be made) and latency (the time between the very start of image capture and the reception of the predicted gaze location in the application). Because we configure our algorithm to take images and predict as soon as it has completed the previous prediction in a while loop, the latency of our predictor is simply the multiplicative inverse of the throughput, plus the network latency. The end-to-end latency of the model will depend on the application's complexity; the overall latency is given by

$$T_{KNN} + T_{network} + T_{app}.$$

5. Experimental Results

We performed error analysis over different model configurations using a similar program as was used for calibration, for convenience. In order to qualitatively test our predictor, we also modified the code for Assignment 3 (foveated and depth of field rendering) The results for various configurations are summarized in Table 1. We measured the throughput and latency using timestamps printed at each iteration of the predictor, and at each reception of gaze data at the application (neglecting the application's contribution to latency). We compared these metrics between various model configurations and Krafka et. al's iTracker (10-15fps) and summarized them in Table 2.

Nimg	Ncal	PCA_{type}	N_{PCA}	k	μ	R	ϵ
175	25	0	–	1	1	591	5.33
175	25	0	–	4	0	511	4.61
175	25	0	–	8	0	506	4.57
175	25	0	–	20	1	437	3.95
175	25	1	40	4	1	567	5.12
175	25	1	40	8	0	529	4.78
175	25	1	80	8	1	529	4.78
175	25	1	80	4	1	567	5.12
648	36	1	40	1	1	363	3.29
648	36	0	–	8	1	325	2.94
648	36	0	–	4	0	326	2.95
648	36	1	40	8	1	329	2.98
648	36	1	80	8	1	329	2.97
648	36	1	40	1	1	363	3.29
648	36	1	40	20	1	318	2.87
648	36	1	80	20	1	318	2.87
648	36	1	40	80	1	320	2.90
648	36	1	80	80	1	333	3.01

Figure 9. Quantitative results for different models. R is the root mean square error in pixels, and ϵ is the angular error in degrees based on a viewing distance of 50cm.

Nimg	Ncal	PCA_{type}	N_{PCA}	k	T	fps
175	25	1	40	1	.1269	7.87
175	25	0	–	4	.343	2.90
175	25	0	–	8	.340	2.93
175	25	1	40	4	.1262	7.91
175	25	1	40	8	.1255	7.96
175	25	1	80	4	.1846	5.41
175	25	1	80	8	.1844	5.42
648	36	0	–	4	1.12	0.89
648	36	0	–	4	1.09	.91
648	36	0	–	80	1.12	.88
648	36	1	40	1	.2789	3.58
648	36	1	40	8	.2612	3.82
648	36	1	80	8	.4687	2.13
648	36	1	80	20	.4645	2.15
648	36	1	40	20	.2783	3.59
648	36	1	40	80	.27560	3.62

Figure 10. Latency and Throughput for various configurations. T is the latency in seconds. We found that most parameters did not affect the latency.

6. Discussion and Limitations

The qualitative accuracy of our early models revealed several variables that the predictor was very sensitive to but which we did not carefully control for. One of these was illumination, which heavily influences the performance

of the Hough circle pupil tracker and also the Haar Cascade classifier. A lack of adequate illumination sometimes caused the Haar Cascade classifier to be unable to localize the eye; variations in illumination between consecutive captures may have also caused train set images for different locations on the screen to be similar due to overall similar brightness.

Another factor that led to difficulties with eye center localization was the use of the iris to perform circle detection instead of the pupil itself. This was mainly due to the fact that we were working with visible light illumination under which the pupil is difficult to localize. The iris is often occluded by the eyelids and sides of the eye during eye movements which makes circle detection unreliable. This was the main reason why our eye center tracking was too unreliable and inaccurate, forcing us to opt for a non-geometric approach to gaze tracking

Another factor was head orientation and head position relative to the screen. A rotation or translation of the head during the calibration process, or a difference in position between the calibration and application caused the predictor to perform poorly. Without a rig to fix the location of the user, our system would not likely be robust to natural movements of users during regular use. The overall stability of the mechanical setup, being a prototype, was not ideal and possibly added extra challenges for the predictor.

We also found that the effectiveness of the algorithm was bottlenecked significantly by the limitations of the Raspberry Pi. Because the training data must be loaded into RAM while the training set is being constructed, and while the predictor is running, we had to cut down the size of the saved images for the train set in order to have a reasonable number of images available at each calibration point, which reduced the available search space. In addition, it reduced the quality of the eye images, some of which were slightly cut off, as is visible in Figure 8.

The sizes of the images also affected the throughput of the algorithm. Given the complexity of PCA and the amount of computation required to compute the distance between fairly large matrices, the worst-case throughput with minimal image compression was less than 1 Hz. We found that image compression significantly improved throughput, but it is possible that the compression removed features that may have been helpful for the classification.

In addition, we avoided using PCA for many of our tests because the performance did not seem to improve and the time required to compile the training set; this may have hurt the robustness of our algorithm to variations in illumination between calibration and application. PCA is particularly useful not just because it compresses the photos, but also because it helps reveal the features which are most important in the data. Additionally, images must be standardized (where the mean of the image is subtracted from all pixels) before PCA, which also improves robustness against variations in brightness.

In our tests, we found that a large k tended to improve the accuracy of the model only up to a point because a sufficiently large k, particularly when $k > \frac{N_{img}}{N_{cal}}$, would force the model to include distant neighbors in the gaze prediction. This sometimes manifested in a jitter in the qualitative test, where the estimated gaze would oscillate between two points and sometimes cause an uncomfortable change in depth rendering. In addition, a weighted average did not appear to make a significant difference in the effectiveness of the algorithm, which suggests that the differences between the images due to the eye orientation at different calibration points is not significant enough compared to the noise.

We found that PCA improved the throughput of the algorithm by reducing the size of the inputs to the k nearest neighbor algorithm. A larger number of images generally did better in terms of accuracy but this came at a cost in throughput and latency. PCA was all but required to make these models with larger training sets viable.

7. Future Work

Our primary future work is porting over our system onto an HMD, specifically the Viewmaster HMD. Implementing our system on the HMD poses particular challenges but also alleviates some of the issue we have with our current system.

In terms of challenges, porting our system onto an HMD would pose difficulties in mounting the camera such that it faced the eye straight on without causing discomfort or considerable view occlusion. Furthermore, we predict that we may have issues with illumination intensity and variation since almost all the illumination for an HMD system would come from the display. With an HMD system however, we would not have problems with head movement since the screen mirrors the head movement. Furthermore, the eye to screen distance would be fixed making calibration easier and more accurate.

We also want to investigate eye center tracking using IR illumination. Most commercial eye trackers make use of IR illumination since the pupil can be far more easily localized under IR illumination since it reflects IR light. With IR illumination, we would be able to solve many of the challenges we faced such as variations in illumination (with a constant brightness IR LED for illumination) and inaccurate eye-center tracking (the pupil is far less occluded by the surrounding eye even during large off-center eye shifts). We opted not to use eye-tracking for this project due to concerns over damaging user's retinas since IR illumination must be done carefully for it to be safe.

8. Conclusion

We created a system for performing gaze tracking using a Raspberry Pi and camera mounted onto glasses frames. We were able to achieve an angular error of $\deg 2.87$ with a throughput of 3.59fps and a latency of .278s. The model parameters for that are given by:

kNN*_params = $[648, 36, 200, 200, 40, 1, 20, 1, 1.3, 5]$

The total cost of the system not including the computer used for rendering the graphics was around $70, which is around 10 times cheaper than the cheapest commercial eye tracking systems. Of course, our system achieves nowhere near the accuracy or throughput as the commercial systems. However, we conclude that with slight improvements such as the use of IR illumination, HMD integration and a more sophisticated k-nearest neighbors implementation we would be able to achieve far greater accuracy. Therefore, the real limiting factor would be the throughput. However, with IR illumination we believe that we would be able to work with lower resolution images without sacrificing accuracy and therefore push up the throughput as well.

9. Acknowledgements

We thank the EE267 teaching team for their support throughout the quarter and for purchasing some of the hardware used for this project. The author thanks Jananan Mithrakumar (4-unit student; project partner) for feedback on the report and help at all stages of the project.

References

[1] H. Chennamma and X. Yuan. A survey of eye-gaze tracking techniques. *Indian Journal of Computer Science and Engineering*, 4(5):388–393, 2013.

[2] D. Dunn. Required accuracy of gaze tracking for varifocal displays, 2019. IEEE VR Conference 2019 Submission.

[3] D. W. Hansen and Q. Ji. In the eye of the beholder: A survey of models for eyes and gaze. *IEEE transactions on pattern analysis and machine intelligence*, 32(3):4478–500, 2010.

[4] W. Heide et al. Electrooculography: technical standards and applications. *Electroencephalography and clinical neurophysiology. Supplement*, 52(1):223–240, 1999.

[5] D. M. Kimmel, Daniel L. and W. T. Newsome. Tracking the eye non-invasively: simultaneous comparison of the scleral search coil and optical tracking techniques in the macaque monkey. *Frontiers in behavioral neuroscience*, 6(49), 2012.

[6] K. Krafka et al. Eye tracking for everyone. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[7] E. C. Lee et al. Robust gaze tracking method for stereoscopic virtual reality systems. *International Conference on Human-Computer Interaction*, 2007.

[8] J. Lemley et al. Efficient cnn implementation for eye-gaze estimation on low-power/low-quality consumer imaging systems. *arXiv preprint*, 2018.

[9] Sesma et al. Evaluation of pupil center-eye corner vector for gaze estimation using a web cam. *ACM Proceedings of the symposium on eye tracking research and applications*, 2012.

[10] J. N. van der Geest and M. A. Frens. Recording eye movements with video-oculography and scleral search coils: a direct comparison of two methods. *Journal of Neuroscience Methods*, 114(2):185–195, 2002.

[11] C. M. Yilmaz and C. Kose. Local binary pattern histogram features for on-screen eye-gaze direction estimation and a comparison of appearance based methods. *International Conference on Telecommunications and Signal Processing (TSP)*, 2016.