Eye Gaze Tracking

Trinity College Engineering Department

By: Giselle Garcia, Khari-Elijah Jarrett, Mauricio Uyaguari

Faculty Advisor: Professor Ning

Table of Contents

I. ABSTRACT

Undeniably, the computer has become a fundamental element of modern society. We are now so immersed in the Digital Age that we have become dependent on computer technology in all aspects of life. Unfortunately, this dependence has created an unavoidable division among society whereby those who suffer from physical impairments lack accessibility to the computer's standard modes of input. This project aims to assist those with limited mobility of their hands by providing a hands-free alternative to using the computer. We plan to achieve this by developing a non-intrusive human-machine interface by means of gaze-tracking whereby we will determine where the user is looking and display the results. We will use strategically placed infrared light emitting diodes to produce corneal glints from the user's eyes. Image processing technology will be utilized to map the location of the glints with respect to a chosen physical landmark of the human eye (eg. the pupil) onto the corresponding point of gaze on the monitor. The components of this project include: the design of the vision subsystem to capture images of the user's eye, the image processing algorithm to detect the glint and determine its location, the correlation matrix transformation algorithm, development of a user friendly GUI for calibration, operation and verification as well as experimental tests for performance analysis.

II. PROBLEM DEFNITION

To design an inexpensive, non-invasive eye gaze tracking system that outputs where the user is looking with respect to the computer screen for simple input commands. This will be achieved without the use of the user's hands. Our system is based on the Gaze Controlled Human-Computer Interface project that was previously done by a group of Trinity students in 2012. Our project is founded on their theory but we use a different method to finding the center of the pupil and the location of the four glints.

III. INTRODUCTION

It is important to be familiar with the human eye anatomy to fully understand how our method of eye gaze tracking works. The figure below highlights some key feature of the human eye.

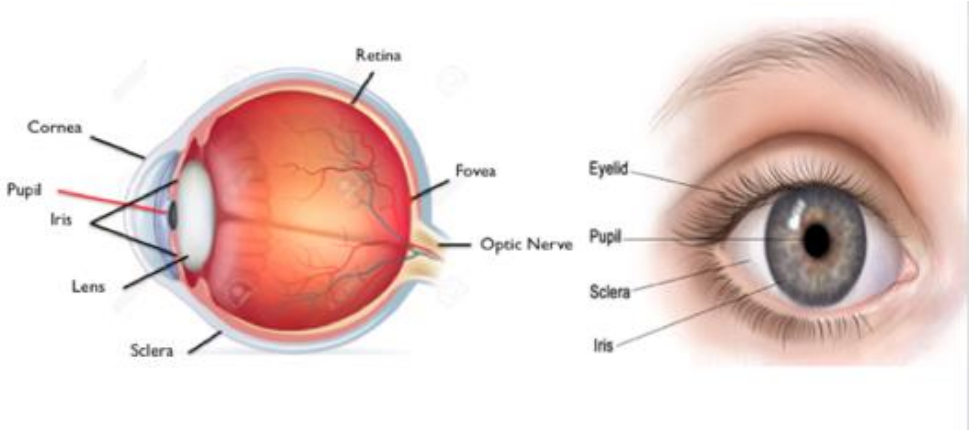

Figure 1 Annotated Human Eye

Light travels through the human eye via two axes – the optical axis and the visual axis. The optical axis is a straight line that passes through the center of the cornea, pupil, lens and retina – it is orthogonal to the cornea. The visual axis is a straight line that passes through the center of the pupil to the fovea.

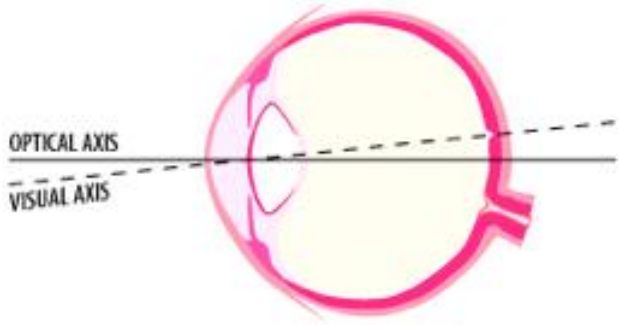

Figure 2 Visual and Optical Axis in Human Eye

The line of sight and the visual axis are functionally the same. Consequently, if we want to determine where the person is looking i.e. their line of sight, we can measure the visual axis of their eye. However, the visual axis varies from subject to subject therefore it has limited use in a system that is intended for multiple people. Instead, the optical axis, which remains constant among subjects, can be used as a good approximation of the visual axis.

The optical axis, as stated before, passes directly through the center of the pupil. As a result, the center of the pupil can be used to measure the optical axis i.e. approximate the visual axis and thus determine where the user is looking.

In order to determine where the user is looking on the computer screen, however, a second piece of information is needed in addition to the location of the pupil center- the person's position in relation to the monitor. According to Yoo et Al, an effective way to create a reference to the monitor is to place four LEDs at the four corners of the computer monitor as seen in Fig.3 below.

Figure 3 LED Placement for Yoo's Method

The LEDs are reflected in the user's cornea. Fig. 4 below shows an example of the four glint reflections on a user's eye if the user is looking directly at the monitor.

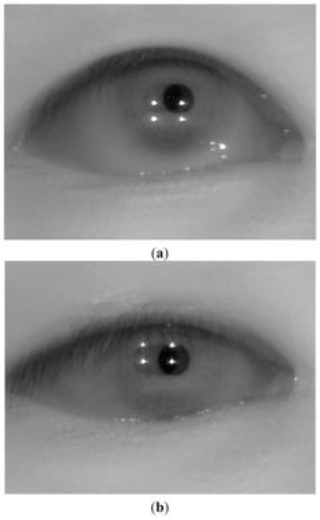

Figure 4 Glint Reflections in User's Eye

In Yoo's method, it is assumed that the user's cornea is a flat surface and that it reflects like a planar mirror. With this assumption, the relative the location of the pupil center with respect to the box formed by the four corneal glints is the same as the relative location of the person's point of gaze with respect to the box formed by the four LEDs on the monitor. This relationship is known as the Cross Ratio approach.

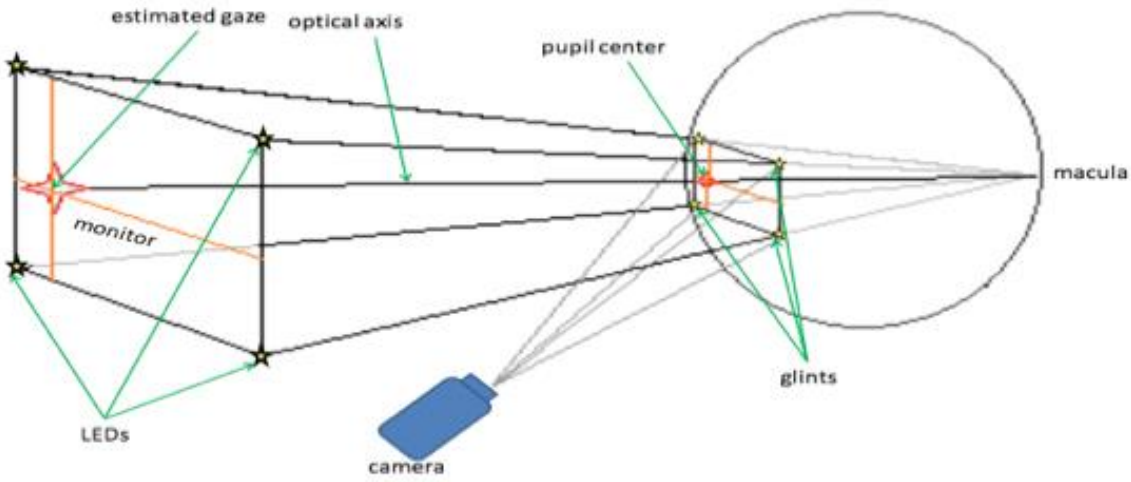

Figure 5 Relationship Between Point of Gaze and Pupil Center

In other words, the box formed by the corneal glints on the user's eye is a linearly scaled version of the box formed by the LEDs on the monitor. Consequently, the location of the pupil center with respect to the four reflected glints represents a scaled version of the user's point of gaze

with respect to the four LEDs on the monitor. Using this information, we can detect a user's point of gaze on the monitor by first determining the pupil's location to the four corneal glints.

IV. PROCEDURE

Our approach to creating the real time eye gaze tracking system was divided into specific milestones:

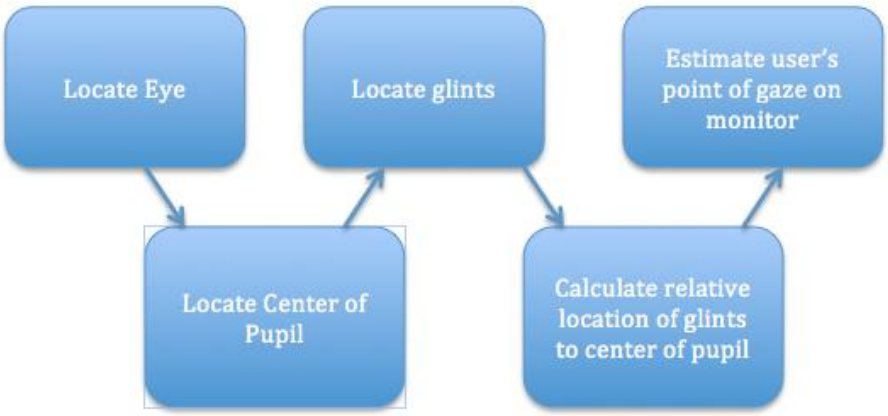

Figure 5: System Procedure

1. Location of Eye

Using a picture of a person's face taken from a webcam, develop an algorithm that would extract only the eye from the picture i.e crop out the rest of the face.

2. Location of Center of the Pupil

With the picture of just the eye obtained from Step 1 above, develop an algorithm to locate the center of the pupil.

3. Location of Glints

Set up monitor with LEDs on each corner and take a picture as in Step 1 with the corneal glint reflections. Using this picture, develop an algorithm that would be able to locate the center of each of the four glints.

4. Calculation of Relative Location of Pupil to Glints

Develop algorithm to determine the location of the pupil with respect to the polygon formed by connecting the centers of each of the four glints.

5. Estimate User's Point of Gaze on Screen

Determine scalar relationship between box formed by glints and box formed by LEDs on the monitor and use this relationship to estimate the user's point of gaze on the screen using the relative location of the pupil center to the glint reflections.

V. IMPLEMENTATION OF EYE GAZE SYSTEM

Part 1: Initial Testing

Our first step was to locate the eye and the center of the pupil. We decided to use a webcam with a relatively small resolution because the images would take a shorter time to process using MATLAB than a larger resolution image as well as the fact that it was more than sufficient to obtain all the information necessary. We settled on a PC webcam with a resolution of 640x780pixels, which was provided to us by Professor Ning.

The webcam was placed on the table in front of the monitor approximately 6 inches from the eye.

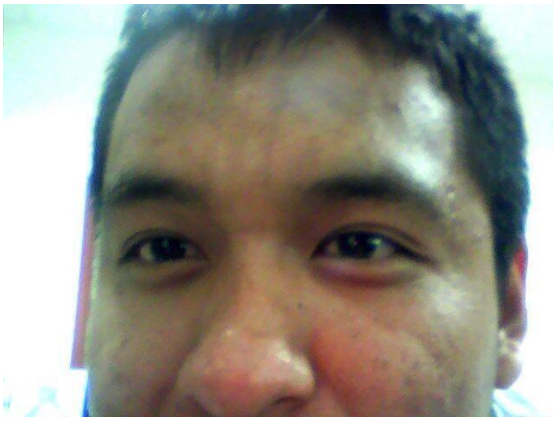

Figure 6: Image obtained from PC Camera 6 inches from the eye

We were able to create an algorithm using these images to obtain locate the eye which is the area of interest on the face as well as the center of the pupil. To locate the eye, we found the 100pixel x 100 pixel box that contained the most amount of white (RGB value = 255) pixels. To locate the center of the pupil, we used MATLAB's image processing saturation function on the image of the eye obtained in the above step. This function uses the intensity (intensity scale:1 = white, 0 = black) of the color and scales it from white to black with black being the most intense.

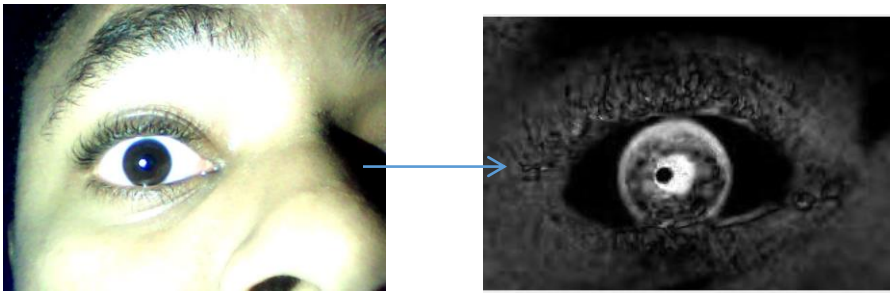

Figure 7: Before and After Saturation of Image

The pupil which is the white circle in the middle of Fig. 7 (after saturation) was then using the Slit Method explained in the Software section.

Part IIa: LEDs

The next step was to obtain the locations of four LED glints in the user's eye. We started with one 5mm round red LED connected to a 330ohm resistor and a 5V supply from a Cadet board. The LED was taped to the top right corner of the monitor and pointed towards the user's face.

We ran into our first problem here. The LED glint was not bright enough to show up in the captured image at the desired distance. We were only able to obtain images of the glint when the webcam was approximately 3 inches from the user's face. This set up was not in agreement with our standards of practicality and user comfort. We experimented with two options – brighter LEDs and IR LEDs.

To obtain a brighter LED light, we tried using a 10mm red LED and also a Super Bright White 10mm LED. Both did not work reliably. Instead, we decided to switch the IR LEDs because of the fact that there will be less noise to interfere with the signal and it provides more comfort for the user as it is invisible to the naked eye. We used 5mm IR leds.

Part IIb: Camera

Our original PC camera was only able to work with visible light. We tried to experiment to remove the IR filter on the lens but we were unsuccessful. There were two options for our camera: to buy a new IR camera or buy the same camera that was used by the previous group that did the experiment (which we know worked by removing the IR filter). We decided to test both options to decide which worked better. We purchased Camera 1: 1.0Megapixel USB camera with IR cut and IR LED for Day and Night Smart Video Surveillance as well as Camera 2: a Logitech QuickCam Communicate MP, which was used by the previous group. The Logitech Cam, like our initial camera has a built in IR glass filter. This time, we were able to successfully remove the IR filter and add a visible light filter to obtain only IR images. Note that Camera 1, the Smart Video camera gives us an image with both visible and IR light.

Smart Video Cam 4 inches away

Logitech Cam 2 inches away

Figure 8: Images Obtained from Different Cameras

However, the problem still persisted in that the camera was too far away to get a reliable image of the glint. The solution was to use a zoom lens. We had a zoom lens from the previous group available to us. This zoom lens was able to attach to the Logitech camera. This lens solved our problem and was able to give us a reliable LED glint from 12 inches away. This can be seen in the Fig 9 below.

Figure 9: IR LED Reflection from 12 inches away

Now that we were able to obtain the reflection for one glint, we needed to see if we could get reflections for four glints. We connected four IR leds to the four corners of the monitor using tape and pointed them towards the user's face. The LEDs were connected to a 330ohm resistor and the 5V supply from a Cadet board. We were successfully able to obtain an image of the four LED reflections in the user's eye as seen in Fig 10 below.

Figure 10: 4 IR LED Reflections from 12 Inches Away

Once we knew that we were able to obtain usable images with the camera and IR LEDs from an ideal distance away, we proceeded to set up our official monitor that would be used for the experiment. We had a 37.8cm x 30.3cm monitor that the previous group used available to us so we decided to use it. This monitor already had holes in the edges of the monitor casing that would allow for the LEDs to be wired inside the casing and protrude to the outside. We decided to use 8 IR LEDs, two on each corner to create brighter glints and they were all wired in parallel.

A 330ohm resistor was connected to each LED and they were wired to the same ground and 5V wires, which would be externally connected to a Cadet Board.

We ran into another problem here whereby not all four glints would show up 100% of the time. We experimented with different distances for both the webcam from the user and the height of the user's eye. We came up with a final hardware set up from which we were able to obtain the four glints 100% of the time. This set up is shown in Fig. 11 below.

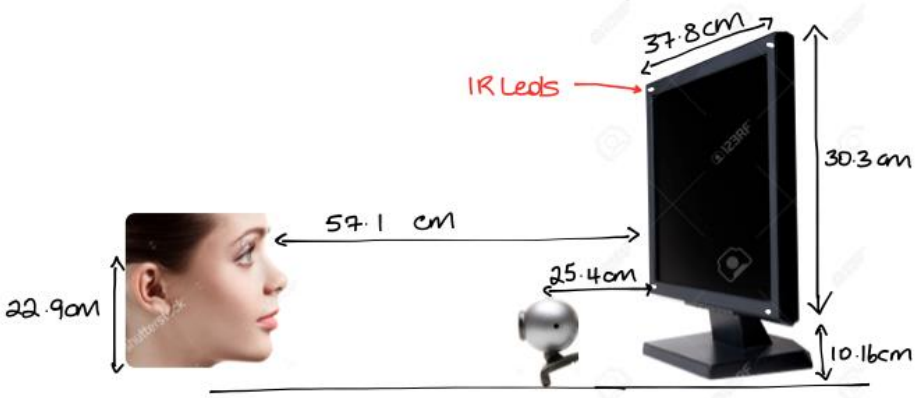

Figure 11: Final Hardware Set Up

Part III: Aesthetics and Improvements

Up to this point, the user's head was not fixed. Instead, we had the user's head resting on a make shift platform and we were just adjusting the webcam manually to find the user's eye. In order to make the design more aesthetically pleasing and robust, we added in a headrest. A wooden headrest was already available to us as it was used by the previous group so we simply cut the legs of it to our desired height.

Part IV: Final Design

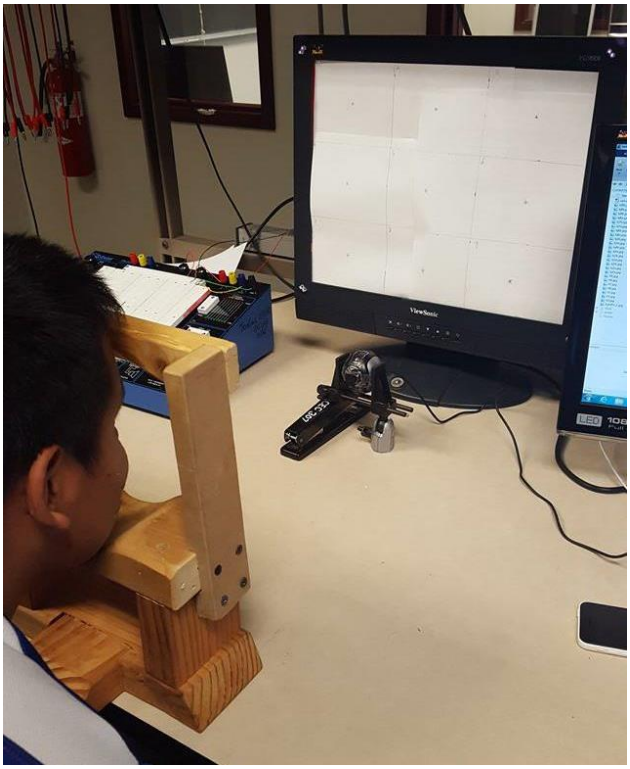

Figure 12: Final Design Set up

Figure 12 above shows our final design set up for the experiment. We use a second computer to analyze the pictures to avoid unwanted light from the monitor. The picture is captured manually by a third party and automatically uploaded to MATLAB. The third party is needed again to run the code on MATLAB. The output is the GUI showing the nine zones with the resulting zone outputted from the code highlighted in green. There is also a voice feedback declaring which zone is the result.

VI. SOFTWARE DESIGN TO TRACK EYE GAZE

Image Processing in MATLAB

In MATLAB, images are represented as three-dimensional matrices. Two of the three dimensions were used to represent the location of the pixel and the third is reserved for the RGB value of the pixel. As a result of the zoom and the camera that was chosen, the Logitech

QuickCam MP, returned images that were 240 x 320 pixels. Furthermore, the amount of red, green, and blue in a pixel were each scaled to a value from 0 – 255. For example, pure green has an RGB value of [0 255 0]. Pure black has an RBG value of [0 0 0]. For the purpose of this project, we will call the sum of the red, green, and blue values the "sum value." The sum value roughly represents the brightness of a pixel. Pure black has a sum value of 0, while pure white has a sum value of $255 * 3 = 765$

$$sum\ value = red\ value + green\ value + blue\ value$$

Interesting Zone

Before taking any other steps, it is imperative to minimize how much of the picture needs to be analyzed. In other words, it is crucial to eliminate any possible noise in the image. For the purposes of this project, any part of the eye that is not the area immediately surrounding the pupil and the glints is considered noise. By eliminating noise, the image becomes availed to more techniques. For example, the brightest part of an entire image may change depending on the image, while the brightest area around the pupil is more likely to be consistent. This consistency can be used to program algorithms to tell us about the image and where the user is looking.

The "zone" was defined as the area of the image immediately surrounding the pupil and the four glints. In order to locate and isolate the zone, it was imperative to find a defining feature that was unique to that area. In this case, the defining trait was the darkness of the pupil. To determine the darkest region, a 15x15 pixel rectangle was initiated in the top left corner of the image. The sum of the sum values of RGB in the rectangle was calculated. This process was iterated repeatedly, moving the rectangle five pixels down each time. When the pixel reached the bottom, it started at the top again, five pixels to the right of where it began. This process was repeated until the entire surface of the image was covered. The rectangle with the smallest sum was considered a part of the pupil. The zone was taken to be the 51x51 area with the 15x15 rectangle at its center.

The above algorithm, which was used to find the zone, assumes the pupil will be the darkest part of the pupil. More specifically, the algorithm assumes the darkest 15x15 box in the image will

be a part of the pupil. This proved to be a fair assumption. Furthermore, this algorithm allows for examination of the portion of the image that is of importance to this project.

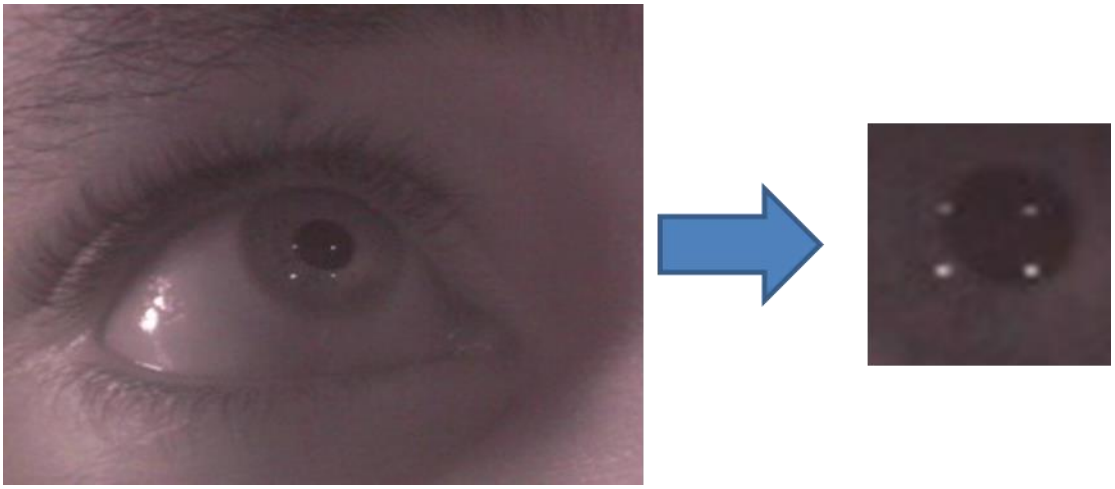

Fig 12. Interesting Zone

Center of Pupil

Before making an attempt to locate the center of the pupil, it was important to determine which portion of the zone is the pupil and which is not. In order to successfully differentiate the pupil, it was imperative to notice one key characteristic of the pupil—its darkness. Even across multiple users, the pupil is noticeably darker than the iris which surrounds it. This characteristic can be used to determine the edge of the pupil. To begin this process, a sliver of the image is considered. This sliver is one pixel wide and is the height of the image. Initially, the sliver is located on the left-most edge of the image. The "sum value" of the pixel is considered the sum of the three values denoting the amount of red, blue, and green in the pixel respectively.

The sliver is moved from left to right until it reaches a point such that more than three pixels in the sliver have sum values less than or equal to X (details on how we find X to follow). Where the sliver stops is considered the left-most edge of the pupil. Then, the same procedure is executed from the right, as well as the top and bottom. After the leftmost, rightmost, top, and bottom locations of the pupil are found, finding the center of the pupil is as simple as taking the average. The left and right averages are taken to find the width-wise location and the top and

bottom averages are taken to find the height-wise location. Thus, the center of the pupil is located.

To find X, the sum value at which the sliver should stop, a 3x3 sample is taken from the center of the image. The "zone" is 51x51 pixels large, so the sample comes from pixels 25-27 x 25-27. Then, the sum value of each of those nine pixels were examined and stored in a 3x3 matrix. If any sum value is greater than 200 (the value at which the pixel is likely a glint or part of the iris), then it is removed from the matrix. If every sum value in the matrix is greater than or equal to 200, the sample is retaken from the 3x3 box immediately above the last one. The average of the remaining sum values in the matrix is taken as X. This value is taken as the approximate sum value of the pupil.

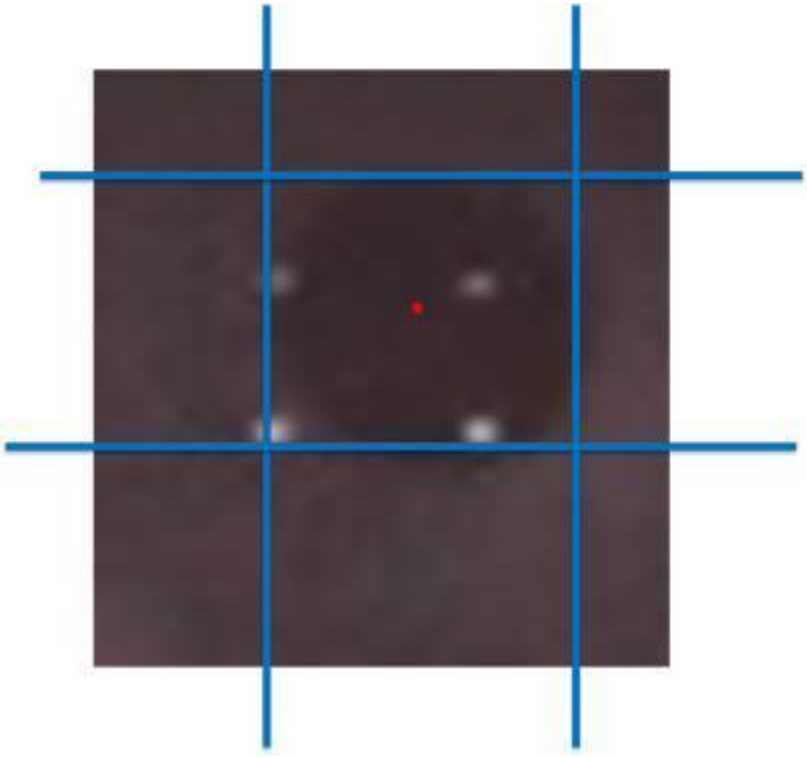

Fig 13. Finding the Center of Pupil: Slit Method

The algorithm to find the center of the pupil makes a few assumptions. The primary assumption the algorithm makes is one about the shape of the pupil. It is assumed that the pupil is approximately a circle. This assumption is reasonable, because even from subject to subject, the

shape of the pupil doesn't change. Also, this algorithm is dependent upon at least a part of the pupil being in the center of the zone or higher. This means the successful execution of this portion of the program relies on successfully finding the zone.

Glint Locations

The next process is detecting the location of the four glints. This would have been difficult using the regular "interesting" zone image because of the excessive noise that it has. It was, however, easy to note the sharp difference between the pupil and the four glints. In order to locate where these differences occur in the zone, edge detection techniques was used.

Edge detection is a valuable technique and MATLAB has a lot of built in edge functions, each using different mathematical algorithms to output the edge in your RGB picture. In this project, the Pewitt method was used. Other methods, such as the Canny, Montage and the Roberts method were examined rigorously but the Pewitt provided the most accurate results.

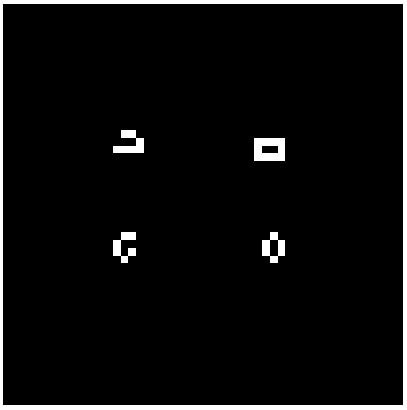

Fig 14. Pewitt Edge Detection Output of Zone

Before running through the Pewitt function, the zone had to be switched to a gray scale image. This was done through the rgd2gray function. This function converts the image by eliminating the hue and saturation. This in other words eliminates the "color" of the image, but the function keeps the luminance so it can be a gray image. After this, the Pewitt edge detection method, scans the image and calculates the gradient at each pixel, and gives the likelihood of each pixel being an edge. The Pewitt function outputs the most likely edge pixels. Ultimately it outputs a matrix of binary numbers, where 1 (logic HIGH) represents an edge and 0 not an edge. This method was tested on 36 zone pictures with four different users. It gave a 94.4% accuracy in

outputting the four glints with minimal noise. The other methods either outputted too much extra noise, or did not detect the glints as edges.

Now, that we have a cleaner version of the zone aimed specifically at locating the glints, we can develop the algorithm. The x and y-locations of every 1 value (logic HIGH) in the matrix is stored into a table. This table represents all possible locations for glints. From this table, the location of all four glints will be determined. Allow G1, G2, G3, and G4 to denote the locations of the top-left, bottom-left, top-right, and bottom-right glints respectively. To find G1, program searches through the table of possible glint locations and selects the one with the lowest x + y value. The glint with the lowest sum of coordinates is the top-left most glint since the origin is the top-left corner of the zone.

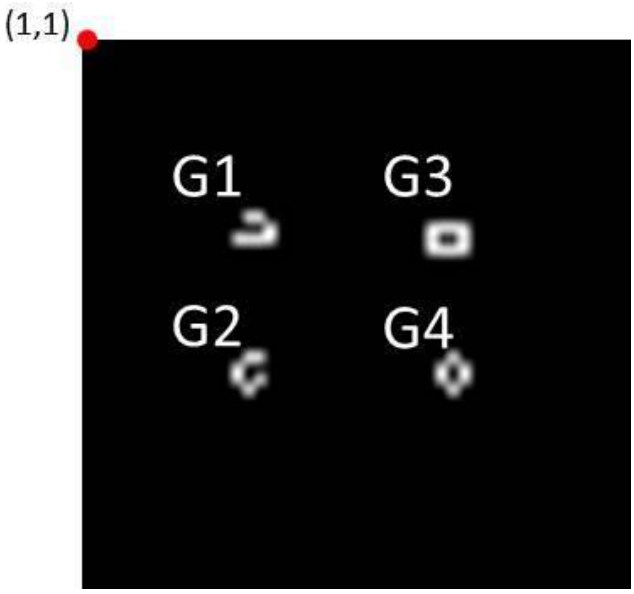

Fig 15. Glints through Edge Detection

To find G2, the location of G1 is used. Let the location of G1 be denoted, (x_1, y_1), where x_1 is the x-value and y_1 is the y-value. Then, any location in the table with an x-value between $x_1 - 2$ and $x_1 + 2$ is considered. Visually, this means a vertical tube 4 pixels wide is considered. The location with the largest y-value is considered G2. Visually, this is the southernmost pixel in the table. Similarly, a horizontal tube is used from G1 to find G3. Any location in the table with a y-value between $y_1 - 2$ and $y_1 + 2$ are considered. Once values for G2 and G3 are established, a

horizontal tube from G2 and a vertical tube from G3 are considered. A pixel is considered G4 if and only if it lies in the intersection of those two tubes.

Calculating Gaze/GUI

The four glint locations and the center of pupil location have now been established. Using the four glint locations a box is constructed. This box will have a height and a length as defined below.

Let G_i's location be defined as (x_i, y_i) for glints 1,2,3,4.

Box Height = $(\frac{y_2+y_4}{2})$-$(\frac{y_1+y_3}{2})$

Box Length = $(\frac{x_3+x_1}{2})$-$(\frac{x_1+x_2}{2})$

Next we take *x* and *y* difference between the center of the pupil and *G1*. These distances will be called *xdistance* and *ydistance*, respectively. The ultimate goal is to show which grid in the 3x3 monitor the user is looking. To do this, the box height and box length was divided into three parts each. Thus this created a virtual 3x3 grid. The program used the *xdistance* to determine which column the user was looking at. The *ydistance* was used to determine which row the user was looking at. Using this, the grid number that the user was looking at was outputted.

After determining where the user was looking the next step was providing feedback to the user. This was done by a Graphic User Interface (GUI). MATLAB has the uicontrol functions which lets you construct a GUI, and add special features to it. Although MATLAB, has a user friendly like program called GUIDE to make it easier to construct a GUI, building the GUI from scratch made it easier to make changes further on. The GUIDE method would have restricted our GUI after it was initially constructed.

Thus, a 3 by3 grid was constructed with nine boxes labeled. When the program calculated that the user was looking at a specific grid, the GUI would highlight this grid in light green as shown below. A special feature was also added for the GUI to tell the user where he is looking through sound. This was done so more people are able to fully use this program.

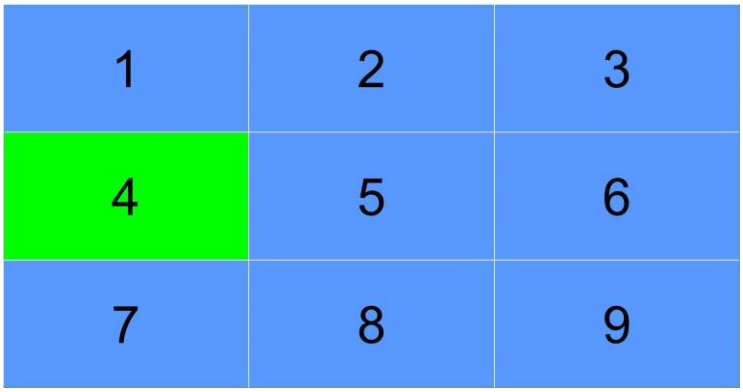

Fig 16. Graphic User Interface

VII. DESIGN CONSTRAINTS

A big constraint for this project was monetary. As a group, we received $300 dollars from the engineering department, which would limit us on the material we bought. Additionally one of our main goals was to produce a low cost system in order for it to be more affordable for lower income people. This constraint came up constantly in our design project. In our case, a lot of the equipment such as the monitor and zoom lens were already provided for us so we did not have to purchase them.

Another major constraint was our technical expertise. When coming up with alternative designs we had to measure our abilities. The group had two electrical engineers and a computer engineer. This gave the group an edge on circuitry and MATLAB but not on mechanical aspects. Thus since there was a learning curve to every design process we considered, we choose one that allowed us to use our strengths rather than a more mechanical concentrated design.

Furthermore image processing requires a lot of complicated mathematical algorithms to analysis some of the images. For example, we tried to come up with an edge detection algorithm catered to our project. After various trials and errors, the built in edge detection functions in MATLAB proved to be more effective than any algorithm we constructed. They still produced some error which given more mathematical expertise we might have been able to fix within our code.

Finally, two constraints in terms of setup are head placement and lighting. Head placement is important as the program relies on static pictures of certain part of your face including your whole eye error but excluding parts like your mouth, your other eye, too much hair, etc. Lighting

also has to be strict since it could affect how the picture comes out and how both the location of the pupil and the location of the glints are calculated. Too much lighting can create too much noise, which could interfere with detecting the glints and the center of pupil. Not enough lighting also complicates our system, since it makes it harder to differentiate between the pupil and iris.

Since, the system aims to be available to as many people as possible; it needs to meet certain engineering standards. These standards include the Occupational Safety and Health Administration (OSHA), Institute of Electrical and Electronics Engineers Standards Association (IEEE-SA) and the Food and Drug Administration (FDA).

VIII. COST ANALYSIS

One of our main goals was to design an inexpensive system so that more people could use it. The cost of this project was separated into two tables. Table A shows the total cost of our final design. Table B shows the total cost of the design process including many parts we ended up not using. The final goal was met, as the final design total came to be $21.21 which is very cheap compared to most of the advanced assistive technology for computer use. Of course, our technology is not complete in terms of providing automatic real time feedback but we estimate this cost to be no more than an additional $25 for an Arduino microprocessor and related circuit components. The inclusive technology company has an eye gazing system for the classroom which costs $2599 (Inclusive Technology). This highlights the cost of many of the technology currently in the market.

Table A: Cost of Final Design

Component	Quantity	Cost
Philmore 5mm Infrared Emiting Diode	8	$11.96
Logitech Quick Cam STX	1	$4.25
Miscellaneous (Wires, resistors, etc…)		$5.00
Total		$21.21

Table B: Cost for Final Design and Design Process

Component	Quantity	Cost
Philmore 5mm Infrared Emiting Diode	20	$29.90
IR Camera	1	$24.99
Logitech Quick Cam MP	1	$24.00
Logitech Quick Cam STX	1	$4.25
LED 875nm Jameco Part #1950622	8	$10.00
Miscellaneous Parts circuit components	1	$15.00
Total		$108.14

IX. RESULTS

In testing our system, the aim was to first create an ideal environment that could make it easier for our code to extract the needed information. Different users, lightning, head placement, camera distance were attempt and an ideal situation was established. The images for this ideal situation can be seen below.

Ideal Images of User looking at 3x3 Grid

For these images, we got a 100% percent success rate. This was due to the changes made in the conditions and the code based on a lot of trial and error. After achieving perfect results from our ideal conditions, our system was tested on different users. Our aim was to test users with distinct eye characteristics. We found users with blue iris, bigger eyes, and bigger pupils at the time. The results can be seen below. Each user has a 3x3 grid below them with a check for success in successfully estimating the user's gaze in that grid. Note that some of the users were tested during different daylight conditions and this could produce some of the errors we received.

Table C: Eye Gaze Results for 8 users[1]

Volunteer 1			Volunteer 2			Volunteer 3			Volunteer 4		
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Volunteer 5			Volunteer 6			Volunteer 7			Volunteer 8		
✓	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
✓	✓	X	✓	X	X	✓	✓	X	X	X	✓
✓	✓	X	✓	✓	X	X	X	X	✓	✓	✓

Twelve of these resulted in error, so currently our program has an accuracy of 83.3%. There seem to be errors within the last column (zones 3, 6,9). We believe this error comes from a large light source to the right of our setup. This light source is more concentrated then the other light sources in the room (aside from the IR LEDs). As a result, a reflection of this light source appears in the user's eye occasionally. This reflection occasionally causes problems with the edge detection portion of the software, which in turn affects the output for glint locations. Additionally some of the errors were very small and our output was on the borderline of two

[1] Thanks to Khari Jarrett, Segun Ajayi , Jake Mevorach, Aaron Porter, Mauricio Uyaguari , Robin Chen, Kathleen Meersman, Griffin Foley for volunteering

boxes. This sometimes occurred in the *x* or *y* direction, which is why it often caused one or more box within a column or row to malfunction.

X. FUTURE WORK AND IMPROVEMENTS

As of now, our system requires the image to be captured manually and the MATLAB code has to be manually run by a 3rd party. Ideally we want a system that is automatic and operated in real time. This would involve adding a microprocessor that would be able to constantly capture images from the camera and automatically run the code. This process would most likely involve further calibration.

Additionally, our code requires adequate IR illumination to operate. As of now, adequate sunlight or else our code will not be able to give consistently reliable results. We need to improve our system so that we can add sufficient IR illumination that would not be dependent on sunlight.

Ideally, this is meant for real time use of a computer as a computer mouse substitute. We could add in a segment that records the user's previous point of gaze and incorporates this in a logical decision to determine where the user is looking now.

Using the current setup, some real world applications that could incorporate our system is using it for online games such as tic-tac-toe and pac-man. Tic-Tac-Toe would allow the system to be tested in real time and would be an ideal game since it is already played on a 3x3 grid. Furthermore, Pac-man would only require a 2x2 grid to play which would reduce errors. This system could be constantly improved, to increase control of the computer in a hands-free manner.

XI. REFERENCES

Huang, Jia-Bin, Qin Cai, Zicheng Lui, Narendra Ahuja, and Zhengyou Zhang. "Towards Accurate and Robust Cross Ratio Gaze Trackers Through Learning From Simulation." University of Illinois,

Huang, Kevin, Petkovsek, Steven, Poudel, Binay. Gaze Controlled Human-Computer Interface. 2012.

Human Vision - Basic Color Theory for the Desktop - Technical Guides. Web. *http://dba.med.sc.edu/price/irf/Adobe_tg/color/vision.html*

Inclusive Technology http://www.inclusive.co.uk/all-in-one-eyegaze-p7299

Urbana Champaign, 2014. Web. *http://research.microsoft.com/pubs/212413/ETRA2014-075-huang.pdf*

XII. Appendix

A. Engineering Standards

The Engineering standards met include the Occupational Safety and Health Administration (OSHA), Institute of Electrical and Electronics Engineers Standards Association (IEEE-SA) and the Food and Drug Administration (FDA).

Occupational Safety and Health Administration (OSHA): The system aims to be available for a huge pool of people for different uses which include many occupation fields. Thus this device, should comply with OSHA standards.

The Institute of Electrical and Electronics Engineers Standards Association (IEEE-SA): This system contains circuitry with the IR LEDs and in the future will probably incorporate more advanced circuitry for real time application. IEEE-SA is important for this system because it aims to hit the international market, and thus this global and recognized standard will help achieve this.

Food and Drug Administration (FDA): The use of this design will probably have some medical applications, and thus compliance with the FDA health regulations is necessary. These regulations could aim toward radiation from the IR diodes.

B. MATLAB Code

Table of Contents


```
%Eye Gaze Tracking Code
%Giselle Garcia, Khari Jarrett and Mauricio Uyaguari
%Advisor: Professor Ning
```

Zone Finding

```
close all;
clear all;

%Reading in the image
PrezFolder = dir('forprez'); %Folder where images will go
Uploadname = PrezFolder(3).name; %take name out of first item in
 folder
UploadPix =fullfile('forprez',Uploadname); %take its "directiion"
b = imread(UploadPix);

zone_size = [15 15];
S = size(b);
height = S(1);
width = S(2);



%Right now this loop iterates through the pic and looks at chunks of
%15*15 size blocks
zone = b(1:1+zone_size(1),1:1+zone_size(2),:); %First box of the image
for i = 1:width-zone_size(2) %i = current location (widthwise)
    for j = 1:height-zone_size(1)%j = current location (heightwise)
            c = b(j:j+zone_size(1),i:i+zone_size(2),:);

            if ( sum(sum(sum(c))) <= sum(sum(sum(zone)))) %Chooses
 darkest box
               zone = c;
               zone_loc = [i j];

            end

    end

end
```

```
w = zone_loc(1)-13;
x = zone_loc(2)-10 ;
zone = b(x :x + 50, w- 5:w+45, :);
imtool(zone) %Show zone
```

Edge Detection

```
gis = rgb2gray(zone);
BWPrewitt = edge(gis,'Prewitt');
BW = BWPrewitt;
```

Finding Glint Locations

```
height = size(BW,1);
width = size(BW,2);
glint_table = [];
glint_thresh = 70;

glint = BW(1,1);
for i=1:width
    for j = 1:height

       if (BW(i,j) == 1)
           glint = BW(i,j);
           glint_loc = [j i];
           glint_table = [glint_table;glint_loc];
       end

    end
end

%Initializing G1 possibilities
G1_table = glint_table(1: ceil(length(glint_table)/2), :);

G2 = [];
G2_table = [];
G3 = [];
G3_table = [];
G4 = [];
G4_table = [];
G4_table2 = [];
G4_table_temp = [];


%Takes G1 as the glint with the smallest sum of coordinates
G1 = glint_table(1,:); %Initializing G1
for row = 1:length(G1_table)
  if sum(G1_table(row,:)) <  sum(G1)
      G1 = G1_table(row,:);
  end
end
```

```
% Fill up possible G2 glints using G1
for row = 2:size(glint_table,1)
    if (G1(1)-2 <= glint_table(row,1)) && (glint_table(row,1) <= G1(1)
 +2) && (glint_table(row,2) > G1(2) + 6)
        G2_table = [G2_table; glint_table(row,:)];
    end
end

% If there are no options for G2
while isempty(G2_table)

    %Delete G1 from the table
    for row = 1:length(G1_table) -1
        if G1_table(row,:) == G1
            G1_table(row,:) = [];
        end
    end

    %Reselect G1 as the glint with the smallest sum of coordinates
    G1 = glint_table(1,:); %Initializing G1
    for row = 1:length(G1_table)
      if sum(G1_table(row,:)) <  sum(G1)
          G1 = G1_table(row,:);
      end
    end

    % Fill up possible G2 glints using G1
    for row = 2:size(glint_table,1)
        if (G1(1)-2 <= glint_table(row,1)) && (glint_table(row,1) <=
 G1(1) +2) && (glint_table(row,2) > G1(2) + 6)
            G2_table = [G2_table; glint_table(row,:)];
        end
    end

end


% Brightest pixel of G2 options is G2
G2 = G2_table(1,:);
for row = 1:size(G2_table,1)
    if sum(BW(G2(2),G2(1),:)) <=
 BW(zone(G2_table(row,2),G2_table(row,1),:))
        G2 = G2_table(row,:);
    end
end


%Fill up possible G3 glints using G1
for row = 2:size(glint_table,1)
    if (G1(2)-5 <= glint_table(row,2)) && (glint_table(row,2) <= G1(2)
 +5) && (glint_table(row,1) > G1(1) + 3)
        G3_table = [G3_table; glint_table(row,:)];
```

```
    end
end

% Brightest pixel of G3 options is G3
G3 = G3_table(1,:);
for row = 1:size(G3_table,1)
    if sum(BW(G3(2),G3(1),:)) <=
 sum(BW(G3_table(row,2),G3_table(row,1),:))
        G3 = G3_table(row,:);
    end
end

% Fill up possible G4 glints using G2
for row = 2:size(glint_table,1)
    if (G2(2)-2 <= glint_table(row,2)) && (glint_table(row,2) <= G2(2)
 +2) && (glint_table(row,1) > G2(1) + 3)
        G4_table = [G4_table; glint_table(row,:)];
    end
end

% Fill up possible G4 glints using G3
for row = 2:size(glint_table,1)
    if (G3(1)-6 <= glint_table(row,1)) && (glint_table(row,1) <=
 G3(1))
        G4_table2 = [G4_table2; glint_table(row,:)];
    end
end

% Takes the intersection of possible G4 glints
if ~isempty(G4_table) && ~isempty(G4_table2)
    G4_table_temp = intersect(G4_table,G4_table2,'rows');

    if ~isempty(G4_table_temp)
        G4_table = G4_table_temp;
    end
end

% In case one G4 table is empty, choose the other
if isempty(G4_table) && ~isempty(G4_table2)
    G4_table = G4_table2;
end

% Brightest pixel of G4 options is G4
G4 = G4_table(1,:);
for row = 1:size(G4_table,1)
    if sum(BW(G4(2),G4(1),:)) <=
 sum(BW(G4_table(row,2),G4_table(row,1),:))
        G4 = G4_table(row,:);
    end
end

% Show glints as green for visibility
zone(G1(2), G1(1), :) = [0 255 0];
zone(G2(2), G2(1), :) = [0 255 0];
```

```
zone(G3(2), G3(1), :) = [0 255 0];
zone(G4(2), G4(1), :) = [0 255 0];
imtool(zone); %Show zone
```

Center of Pupil

```
X = 3; %Amount of pixels in the slit that must meet requirement

% We need to find a threshold value for the COP

% Takes a sample from the middle of the picture
check_zone = 25:27;
mid1 = zone(check_zone, check_zone, :);
mid_box = mid1(:,:,1) + mid1(:,:,2) + mid1(:,:,3);
mid_box = mid_box .* uint8(mid_box < 200);

% If it's too bright try again
while mid_box == [0 0 0; 0 0 0; 0 0 0]
    check_zone = check_zone - 3;
    mid1 = zone(check_zone, check_zone, :);
    mid_box = mid1(:,:,1) + mid1(:,:,2) + mid1(:,:,3);
    mid_box = mid_box .* uint8(mid_box < 200);
end

% Threshold number
cop_threshold = sum(sum(mid_box)) / nnz(mid_box);


% THIS FINDS LEFT EDGE
for i = 5:length(zone)-1
    slit = zone(:,i,:);   %Take vert slit
    temp_slit = slit(:,:,1) + slit(:,:,2) + slit(:,:,3);
    count = 0;

    for j = 1:length(slit)
        if temp_slit(j) < cop_threshold
            count = count + 1;
        end
    end

    if count > X
        leftedge = slit;
        leftloc = i;
        break
    end
end

% THIS FINDS RIGHT EDGE
for i = length(zone)-6:-1:1
    slit = zone(:,i,:);   %Take vert slit
    temp_slit = slit(:,:,1) + slit(:,:,2) + slit(:,:,3);
    count = 0;
```

```
    for j = 1:length(slit)
        if temp_slit(j) < cop_threshold
            count = count + 1;
        end
    end

    if count > X
        rightedge = slit;
        rightloc = i;
        break
    end
end

% THIS FINDS TOP EDGE
for i = 5:size(zone,1)
    slit = zone(i,:,:);   %Take horiz slit and find sum
    temp_slit = slit(:,:,1) + slit(:,:,2) + slit(:,:,3);
    count = 0;

    for j = 1:length(slit)
        if temp_slit(j) < cop_threshold
            count = count + 1;
        end
    end

    if count > X
        topedge = slit;
        toploc = i;
        break
    end
end

% THIS FINDS BOTTOM EDGE
for i = size(zone,1)-6:-1:1
    slit = zone(i,:,:);   %Take horiz slit and find sum
    temp_slit = slit(:,:,1) + slit(:,:,2) + slit(:,:,3);
    count = 0;

    for j = 1:length(slit)
        if temp_slit(j) < cop_threshold
            count = count + 1;
        end
    end

    if count > X
        bottomedge = slit;
        bottomloc = i;
        break
    end
end

% Takes the average of the edges as the COP
centerloc = [(leftloc+rightloc)/2 (toploc+bottomloc)/2];
```

```
% Shows the COP as red for visibility
Cen_ceil2 = ceil(centerloc);
zone(Cen_ceil2(2), Cen_ceil2(1), :) = [255 0 0];
imtool(zone); %Show zone
```

Gaze Approximation

```
BOX = 5; %Initialized guess

% Get a distance from G1 to center of pupil
distance = centerloc - G1;
x_dist = distance(1);
y_dist = distance(2) +2;

% Box height and length
box_height = (G2(2) + G4(2))/2  -  (G1(2) + G3(2))/2;
box_length = ((G3(1) + G4(1))/2  -  (G1(1) + G2(1))/2) ;

% Box approximations
if (x_dist < box_length/3) && (y_dist < box_height/3)
    BOX = 3;

elseif (x_dist >= box_length/3) &&  (x_dist < 2* box_length/3) &&
 (y_dist < box_height/3)
    BOX = 2;

elseif (x_dist >= 2* box_length/3) && (y_dist < box_height/3)
    BOX = 1;

elseif (x_dist < box_length/3) && (y_dist >= box_height/3) && (y_dist
 <  2*box_height/3)
    BOX = 6;

elseif (x_dist >= box_length/3) && (x_dist < 2*box_length/3) &&
 (y_dist >= box_height/3) && (y_dist <  2*box_height/3)
    BOX = 5;

elseif (x_dist >= 2 * box_length/3) && (y_dist >= box_height/3) &&
 (y_dist <  2*box_height/3)
    BOX = 4;

elseif (x_dist < box_length/3) && (y_dist >= 2* box_height/3)
    BOX = 9;

elseif (x_dist >= box_length/3) && (x_dist < 2* box_length/3) &&
 (y_dist >= 2* box_height/3)
    BOX = 8;

else
    BOX = 7;

end
```

GUI

```
% Creates figure
figure('units','normalized','outerposition', [0 0 1 1])


b1 = uicontrol('style','pushbutton',...
                  'string', '1',...
                  'units','normalized',...
                  'fontsize',100,...
                  'BackgroundColor',[.34 0.60 1],...
                  'position' , [0 2/3 1/3 1/3],...
                  'callback',@eg_fun);

b2 = uicontrol('style','pushbutton',...
                  'string', '2',...
                  'units','normalized',...
                   'fontsize',100,...
                  'BackgroundColor',[.34 0.60 1],...
                  'position' , [1/3 2/3 1/3 1/3],...
                  'callback',@eg_fun);

b3 = uicontrol('style','pushbutton',...
                  'string', '3',...
                  'fontsize',100,...
                  'BackgroundColor',[.34 0.60 1],...
                  'units','normalized',...
                  'position' , [2/3 2/3 1/3 1/3],...
                  'callback',@eg_fun);


b4 = uicontrol('style','pushbutton',...
                  'string', '4',...
                   'fontsize',100,...
                  'BackgroundColor',[.34 0.60 1],...
                  'units','normalized',...
                  'position' , [0 1/3 1/3 1/3],...
                  'callback',@eg_fun);

b5 = uicontrol('style','pushbutton',...
                  'string', '5',...
                  'units','normalized',...
                  'fontsize',100,...
                  'BackgroundColor',[.34 0.60 1],...
                  'position' , [1/3 1/3 1/3 1/3],...
                  'callback',@eg_fun);
b6 = uicontrol('style','pushbutton',...
                  'string', '6',...
                  'units','normalized',...
                   'fontsize',100,...
                  'BackgroundColor',[.34 0.60 1],...
                  'position' , [2/3 1/3 1/3 1/3],...
                  'callback',@eg_fun);
```

```
b7 = uicontrol('style','pushbutton',...
                   'string', '7',...
                    'fontsize',100,...
                   'BackgroundColor',[.34 0.60 1],...
                   'units','normalized',...
                   'position' , [0 0 1/3 1/3],...
                   'callback',@eg_fun);

b8 = uicontrol('style','pushbutton',...
                   'string', '8',...
                    'fontsize',100,...
                   'BackgroundColor',[.34 0.60 1],...
                   'units','normalized',...
                   'position' , [1/3 0 1/3 1/3],...
                   'callback',@eg_fun);

b9 = uicontrol('style','pushbutton',...
                   'string', '9',...
                   'fontsize',100,...
                   'BackgroundColor',[.34 0.60 1],...
                   'units','normalized',...
                   'position' , [2/3 0 1/3 1/3],...
                   'callback',@eg_fun);

 box = [b1, b2,b3,b4,b5,b6,b7,b8,b9];

 %sounds for GUI
 [sound1 fs] = audioread('Zone1.m4a') ;
 sound2 = audioread('Zone2.m4a') ;
 sound3 = audioread('Zone3.m4a') ;
 sound4 = audioread('Zone4.m4a') ;
 sound5 = audioread('Zone5.m4a') ;
 sound6 = audioread('Zone6.m4a') ;
 sound7 = audioread('Zone7.m4a') ;
 sound8 = audioread('Zone8.m4a') ;
 sound9 = audioread('Zone9.m4a') ;
% Gui boxes
guibox = BOX;
if guibox == 1;
    set(box,'Background',[.34 0.60 1])
    set(b1,'Background',[0 1 0])
    sound(sound1,fs);
elseif guibox ==2;
    set(box,'Background',[.34 0.60 1])
    set(b2,'Background',[0 1 0])
    sound(sound2,fs);
elseif guibox ==3;
    set(box,'Background',[.34 0.60 1])
    set(b3,'Background',[0 1 0])
    sound(sound3,fs);
elseif guibox ==4;
    set(box,'Background',[.34 0.60 1])
    set(b4,'Background',[0 1 0])
```

```
    sound(sound4,fs);
elseif guibox ==5;
    set(box,'Background',[.34 0.60 1])
    set(b5,'Background',[0 1 0]);
    sound(sound5,fs);
elseif guibox ==6;
    set(box,'Background',[.34 0.60 1])
    set(b6,'Background',[0 1 0])
    sound(sound6,fs);
elseif guibox ==7;
    set(box,'Background',[.34 0.60 1])
    set(b7,'Background',[0 1 0])
    sound(sound7,fs);
elseif guibox ==8;
    set(box,'Background',[.34 0.60 1])
    set(b8,'Background',[0 1 0])
    sound(sound8,fs);
elseif guibox ==9;
    set(box,'Background',[.34 0.60 1])
    set(b9,'Background',[0 1 0])
    sound(sound9,fs);
end

% Move to Used folder
movefile(UploadPix,'Used')
```

Published with MATLAB® R2015a