4-1-2012

# Gaze Controlled Human-Computer Interface

Steve Petkovsek
*Trinity College*, steven.petkovsek@trincoll.edu

Kevin Huang
*Trinity College*, kevin.huang@trincoll.edu

Binay Poudel
*Trinity College*, binay.poudel@trincoll.edu

# GAZE CONTROLLED HUMAN-COMPUTER INTERFACE

SPRING 2012 SEMESTER REPORT

5-7-2012

KEVIN HUANG
STEVEN PETKOVSEK
BINAY POUDEL

FACULTY ADVISOR: PROFESSOR TAIKANG NING

# TABLE OF CONTENTS

## I. INTRODUCTION/MOTIVATION

The personal computer is a ubiquitous tool in modern society. Minimal operation of the personal computer typically requires use of the computer mouse and keyboard. Because of this, people with limited or obstructed mobility of their hands, such as paraplegics, amputees, and sufferers of carpal tunnel or arthritis, may have trouble operating a personal computer. This project aims to take a step towards a novel hands free human-computer interface.

We propose to solve this problem by developing a gaze-tracking interface by which a user can control his/her computer input in a hands-free manner. In our project, infrared light emitting diodes (LEDs) are strategically placed around a computer monitor to produce corneal glints off of the user's eye. Video cameras then focus on these glints, and a customized correlation matrix transformation is used to map the location of the glints with respect to physical landmarks of the human eye (e.g. pupil) onto the corresponding point of gaze on the monitor. The proposed project includes the design of the video vision subsystem to capture images of the user's eye, the image processing algorithm to detect and determine glint location, the correlation matrix transformation algorithm, development of a user-friendly GUI for calibration, operation, and verification, and experimental tests and trials for performance analysis.

The aim of the Human Vision Control Interface is to provide a non-intrusive human-machine interface that allows the user to control the computer using only his or her eye movements as the input. The user will be able to move a cursor into different boxes in a grid on the screen in real-time. This work has great potential in terms of providing opportunities and enjoyment in the work place and at home that may not have been available otherwise.

This project will incorporate novel methods to achieve gaze tracking. By using a probabilistic model (i.e. fuzzy logic) instead of a deterministic one, the accuracy will be greater than in previous iterations of point of gaze technology.

## II. THEORY

### PUPIL CENTER

Several gaze tracking approaches use invasive means to determine the user's gaze location, including transducers placed on the user's head to measure skin electrical potential corresponding to muscle movement, and implementation of contact lens designed exclusively for gaze tracking [1]. This approach is bulky and uncomfortable.

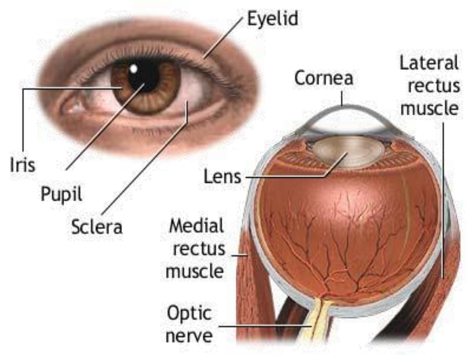


FIGURE I
Human Eye Anatomy [2]

To determine the point of gaze of a human user non-invasively, it is critical to detect key features of the human eye. Figure I. shows the basic anatomy of the human eye. Several physical,

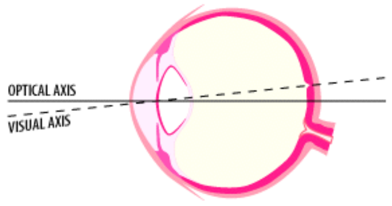


FIGURE II
Visual Axis [3]

outwardly observable components of the human eye include: the iris, pupil, sclera, and eyelid. Of these features, the pupil will be of upmost importance in terms of gaze tracking. This is because the visual axis (a vector that passes through both the macula, or center of the retina, and the center of the cornea) passes through the pupil center. Moreover, the visual axis describes exactly the direction in which a person is looking.

It is difficult to measure precisely the visual axis due to subject to subject variation, however. However, the optical axis (vector passing through the center of the cornea and orthogonal to the cornea) is both a good approximation of the visual axis and consistent from individual to individual.

Both the visual axis and optical axis pass directly through the pupil center, as shown above in Figure II. With that said, observing and locating the pupil center is vital in any effort to non-invasively measure a user's optical axis, approximating the visual axis and thus the point of gaze.

## REFERENCE GLINTS

The location of the pupil center alone is not sufficient to determine a user's gaze location. This is because the problem is three-dimensional and projecting a user's gaze onto a computer monitor is impossible without information regarding the location of the user with respect to the monitor itself. More specifically, detection of the pupil center location with respect to just the user can only provide gaze direction information with reference to the user. The proposed system needs to determine where *on a computer monitor* that the user is looking. Therefore, it is necessary to provide some type of signal to help map the user's pupil location to a location on the computer monitor, a reference signal of some sort. Yoo et al proposed that a simple way to create a reference to the monitor was by placing light emitting diodes (LEDs) within the plane of the monitor [4]. The LEDs produced corneal glints reflected off the user's cornea. The hardware setup and corneal glints from Yoo's experiment are shown below in Figure III.

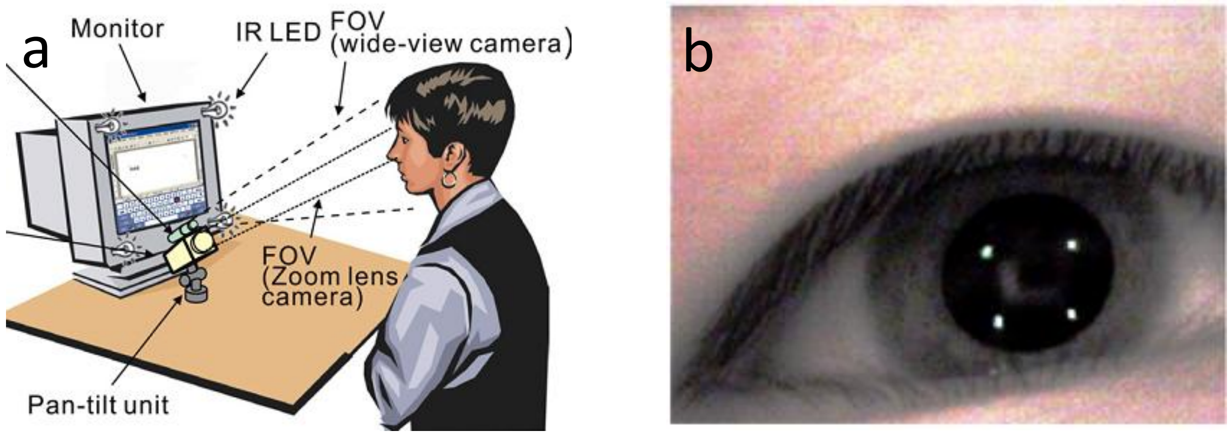


FIGURE III [4]
*a)* Hardware setup for Yoo et al's cross ratio method. *b)*Acquired corneal glints

The method of gaze detection in Yoo's study relies on a basic assumption that the user's cornea can be approximated as a planar surface and reflects analogous to a planar mirror. With this assumption, one can approximate a user's point of gaze on a monitor by computing the relative location of the pupil center to the polygon formed by connecting the four corneal glints. Figure IV shows a simple ray tracing that demonstrates the cross-ratio approximation of gaze location:

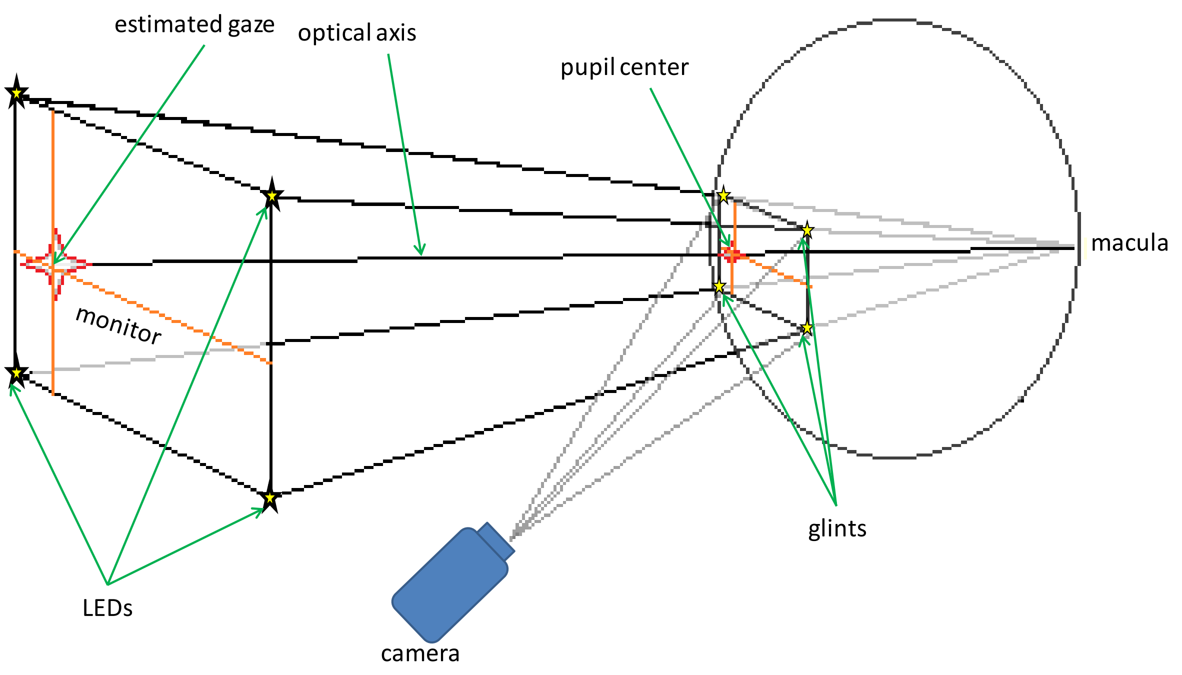


FIGURE IV
Cross-Ratio Methodology

It is observed that by projecting these four glints on the cornea, the entire monitor is projected onto the user's eye (polygon formed by connecting the four glints). Moreover, this projection of the monitor is in the same plane as the pupil, thereby reducing gaze determination to a two dimensional problem. Thus, the relative location of the pupil center to the four corneal glints is an approximation of the intersection of the optical axis and the plane of the monitor, the user's gaze on the computer screen.

Drawbacks for the cross ratio approach include lack of consistent accuracy (deviations of as much as 164.5mm were reported by Kang [5])

## INFRARED (IR) RETINAL REFLECTION

There are two reasons for utilization of IR frequency light as illumination sources. Firstly, IR light is not visible to the human eye (IR constitutes any frequency light with wavelength greater than any visible light, i.e. greater than 740nm). As a result, using IR illumination as glint reflection sources would not distract the user or create unnecessary discomfort as a visible light source could.

Secondly, the human retina acts as a retroreflector of IR light. That is to say that any light entering the pupil will be reflected with minimal scattering and in the same, opposite direction in which it entered. Figure V demonstrates the retroreflectivity of the human retina.

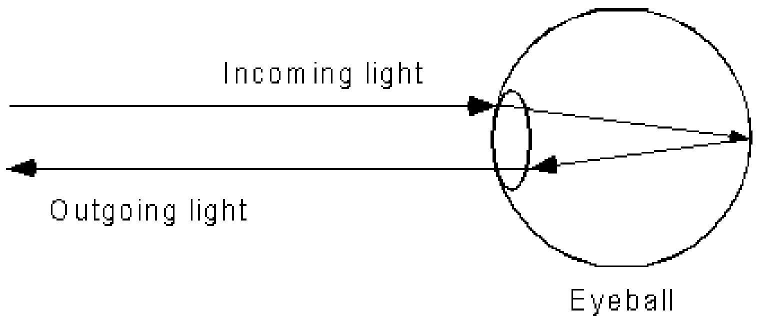


FIGURE V
Human Eye as Spherical Retroreflector of IR

This property of the human retina is useful in terms of detecting the pupil. Acquiring any signature of the retroreflection would require illumination coaxial to the image capturing device. Further, unwanted illumination, such as visible light illumination, which is subject to inconsistency, can be filtered. This preprocessing of the data captured by the camera reduces processing time and guarantees a stable lighting environment that will ensure the required information (i.e. the glint and pupil center locations) will always be extractable.

## III. APPROACH

The gaze controlled human-computer interface team has several technical, personal and other available resources to support the project. First and foremost, the space and tools in Professor Ning's digital signal processing lab are vital to successful project completion. These tools include basic electronic measuring devices and discrete elements, soldering station, computers, basic hardware tools, various FPGA boards, software, cameras and many years of Circuit Cellar, IEEE explore and various other journal articles.

While the bulk of the labor involved is in the field of electrical engineering, several other fields play a significant role. An intimate knowledge of the anatomy and biology of the eye will be required in order to determine exactly where the user is looking and to understand the reflective properties and shape of the cornea. Optics will also be an important physical concept to understand because the system relies heavily on the paths and scattering of light to make its calculations. Additionally, because image capture is a critical component of the system, the knowledge of camera filters and lenses is an integral part to detailed image capture for optimal baseline information. The image processing to be done will require expertise in signal processing. Finally, several computer science methods will be implemented to handle data processing and GUI creation for both the calibration and real-time displays.

In addition, several other resources are available to us directly from Trinity College: the machine shop, wood shop, the EE lab, and tools in the ME lab. These resources will be vital for hardware construction. Furthermore, the team has two professional contacts: Jeffrey B. Mulligan at the NASA Vision group in Moffett Field, California and David Avanesian at the NASA Glenn Research Center at Lewis Field, Ohio have agreed to be professional contacts and advisors throughout the project. Their technical expertise within the field of gaze tracking will be useful throughout the academic year.

## IV. GOALS/CONSTRAINTS

### GOALS

The target system shall improve on previous iterations of POG studies by taking a new approach to determining pupil location. It will provide a non-intrusive environment in which the user can directly manipulate the graphical interface using only eye movements. With these metrics in mind, the most important objectives are as follows:

#### *1. Establish a non-intrusive environment for which POG is suitable:*

Invasive techniques, such as sensors mounted on the face and head, will be avoided. Instead, Head movement and location will initially be restricted, and a headrest will be constructed on which the user must put his chin when using the interface. This prohibitive measure may be relaxed if satisfactory results are obtained from a fixed-head perspective.

#### *2. Detect corneal glints and pupil movements*

An image processing algorithm will be developed to identify corneal glints, the iris, pupil center and sclera region of the user's eye. This will be achieved by analyzing preliminary images of a computer user's eyes captured by a digital camera. Various orientations of both LEDs and camera positioning will be used in this image capture. Following this, image processing algorithms will be tested to isolate visual features consistent throughout the images.

#### *3. Use the cross-ratio method to map eye movements onto the screen:*

When an image feature isolating algorithm is developed, the team will shift its focus on correlating the relative positions of the features to point of gaze, or where on the computer monitor the user is looking. This process will utilize a correlation algorithm using empirically determined statistical quantities and fuzzy logic. The cross-ratio method will be used. At minimum, a 3x3 grid resolution will be achieved.

#### *4. Display Results in a Graphical User Interface*

A user interface will be designed which highlights sectors which are being looked at by the user or somehow informs the user which section of the screen the system believes the user is gazing at. The baseline resolution for such a display is a 3x3 matrix which will be drawn on the screen.

*5. Create an embedded system that will allow real-time application*

The Human Vision Control Interface will use a powerful embedded processor to allow the user's computer to function normally and allow a real-time interaction between the user and the computer, as desired in the final product. Once a reliable, repeatable method of offline gaze estimation is developed, the team will begin to optimize the algorithms and begin programming hardware (field programmable gate array, FPGA). The FPGA will handle the burden of rapid image processing and matrix manipulation.

*6. Expand system functionality through testing*

When the system has been reliably built to function correctly for on individual, the system will be expanded to accommodate a range of users. To do this, students will be invited to test the device and the data acquired will be used to modify our empirical algorithm.

The goals for the Point of Gaze Project can be summarized as follows:

- Build and assemble the hardware necessary for the system (e.g. headrest, camera/monitor hardware)
- Develop an algorithm to detect center of pupil and extrapolate relative gaze location.
- Display output of system on computer monitor. Baseline resolution is a 3x3 display.
- Implement the algorithm in a FPGA board and interface camera with FPGA.
- Test the device using a diverse user group and use the resulting statistical data to determine the accuracy of the system.

## CONSTRAINTS

*1. Technical Expertise*

For the Gaze Controlled Human-Computer Interface the team has researched several possible solutions that involve specialized technical knowledge. This constrains the team regarding which solutions are viable for the amount of time available for the project, as it takes time to learn the specifics for each approach. This also constrains the complexity of the approaches taken to design the system. Further, the team will be creating a complete package of features that allow any individual to use the device without technical expertise. Thus, several areas of computer science, such as GUI development and data structures, will be explored to deliver a polished, user-friendly application.

*2. Mathematical Background*

For Principle Component Analysis (PCA) a high level of classification mathematics knowledge is required to suitably create an image space and cluster images accordingly. The team must be familiar with relevant statistics involved in classifying each image for the approach to be effective. For the cross-ratio approach, the team must be able to develop a function that accurately maps to the screen the measured distances between the pupil and glints in the image.

### *3. FPGA Implementation*

The team planned on working with a DE2 FPGA processor. Each member spent several weeks learning how to use the board properly. Each member took a series of tutorials to learn how to program the desired features in a hardware programing language such as Verilog or VHDL and to interface external devices, such as the camera and local processor, with the board.

### *4. Optics*

This system requires work with both biological optics and physical optics. The team must learn precisely how the human vision system works in order to determine where the user is looking given an image of the user's eye. The team must also be familiar with the geometrical properties of the eye to determine the physical path that the light reflected on the eye is actually taking.

### *5. Practical Constraints*

#### *Gaze Detection Throughput*

The processing power of the FPGA selected will limit the frame rate of the system. If a board without a sufficient number of blocks is selected, the complexity of the algorithm will be limited and the resolution of the camera will be reduced.

#### *Resolution*

The maximum resolution of the system is determined by the resolution of the camera used. If a low resolution camera is used, the resulting resolution of the system may be limited. Considering past projects in this area, the resolution of sectors on the screen may be limited to a 5x5 matrix, though the baseline resolution is a 3x3 matrix.

#### *Head Placement*

The baseline system will rely on a static head position in order to use relative pupil location from a glint as a gaze tracking method.

## ENGINEERING STANDARDS

Food and Drug Administration (FDA) - Since the use of the design can be medical, compliance with FDA health regulations is necessary. This includes radiation from IR diodes, posture, and electrical safety standards.

Occupational Safety and Health Administration (OSHA) – Since the device has potential in the work place, OSHA standards must be complied with.

Restriction of Hazardous Substances (RoHS) – Design will implement Lead-free parts. Lead-free solder will be used for necessary connections.

## V. HARDWARE

### PART I: OFFLINE DATA COLLECTION AND INITIAL TESTING

Based on the goals and the constraints of the project, a hardware setup was designed to collect initial data. The setup consisted of 4 red LEDs, a chin rest, the computer monitor and a camera. Ultimately, the Altera DE2 FPGA board was planned for real-time acquisition and computation of signals in order to calculate a user's point of gaze. The power supply used was a standard voltage power source, and no automation in the hardware was present. The user had to take an image manually using the camera and load it onto the host PC after all desired images had been captured. The images were processed offline using MATLAB. This was done to determine how useful the hardware approach could be and if the required data was useable (see Algorithms, Part 1). The basic hardware setup is shown in Figure VI. Each of these hardware components will be described individually.

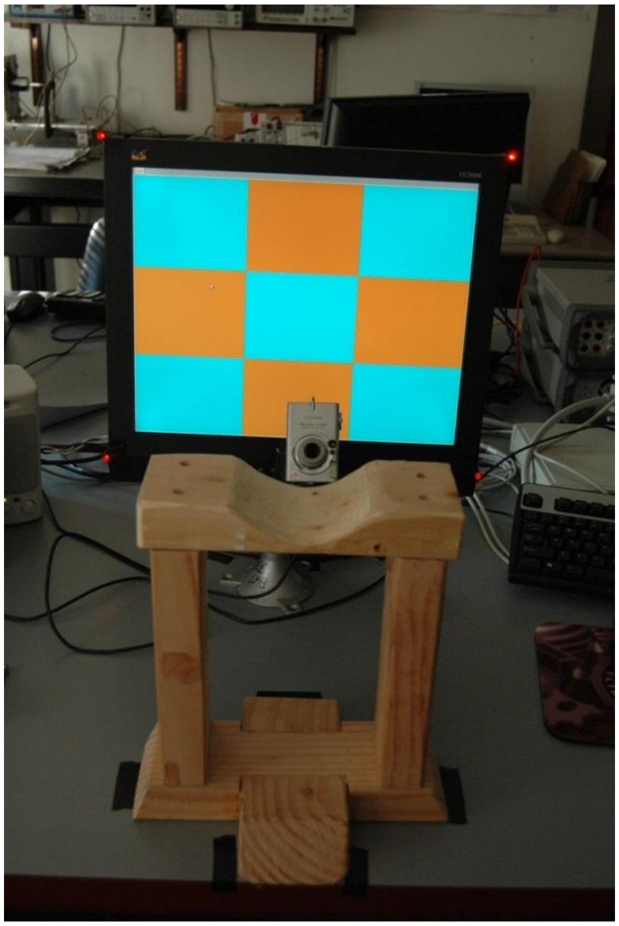


FIGURE VI
Initial Data Collection Hardware Setup

#### LEDS

4 LEDs were placed at the four corners of the monitor to produce glints on the user's cornea. Narrow-angled red LEDs were used in this setup because they were readily available in the lab. The glints produced by these LEDs were bright enough to be captured by a Canon digital camera. The LEDs were supplied with +5V DC voltage through 200Ω current limiting resistors. An alternative approach that was considered was the use of bright LEDs. Bright LEDs are capable of producing high-intensity white light. This would result in a more intense glint (higher signal to noise ratio), but can also produce adverse effects, i.e. discomfort to the user.

#### CHIN REST

A chin rest was constructed out of scrap wood found in Trinity College's wood shop. The purpose of the chin rest is to help a user maintain stationary head placement while using the system. The chin-rest is 25cm tall. Additionally, a forehead rest was added (not shown) to the design to further stabilize a user's head while operating the system. This was added to prevent forward and backward deviation of the user's head placement. To secure the chin rest, simple shop-clamps are used.

### MONITOR

A 19 inch flat screen, liquid crystal display monitor was used in the system. It was placed 35cm from the chin rest to allow a user use the system comfortably. A software-based graphical user interface (GUI) was developed for the project, and divided the screen into a 3x3 grid for testing purposes. The grid itself did not interact with the user but was used as a means to divide the screen into sectors while keeping lighting conditions similar to the final display. To initially test the system, images were taken with the user looking at each grid location and analyzed by the team do determined the primary features that could be extracted from each image (*see Algorithms: Initial Data Acquisition and Analysis*).

### CAMERA

A digital camera was used to acquire images of user's eye (data collection) while looking at different sections of the screen. Pictures of only one of the user's eyes were acquired. This is based on the assumption that both optical axes of the user's two eyes intersect at precisely where the user is looking. This camera was to be replaced with a continuous capture camera for automated data capture.

Various camera positions and magnification magnitudes were tested to find a position that would allow for the best image acquisition for the purpose of this project. These criteria included that the entire (upper to lower eyelid) be captured, and minimal extraneous features included (i.e. nose, eyebrow, etc). It was empirically discovered that placing a zoom camera about 3 quarters of distance between computer screen and chinrest was ideal (approximately 27 cm from the chinrest in our setup). Furthermore, since the user's head was positioned 25cm above the level of the table, it was determined that most users would be looking at a slightly downward angle towards the screen. Thus, the camera was placed near the bottom of the monitor.

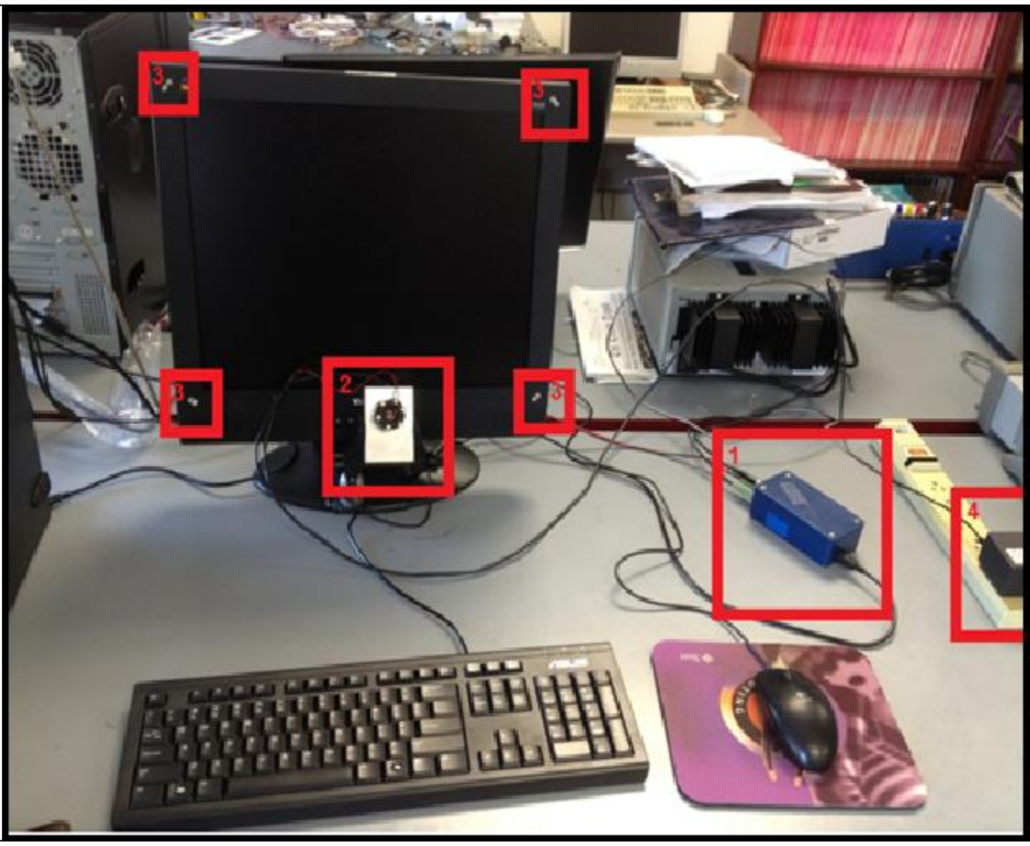
The initial current setup used the Canon Powershot Digital Camera. Different cameras were proposed to replace it. The alternatives included the Terasic D5M camera initially stood out due to its response and sensitivity to the IR spectrum (potential integration of IR illumination) and easy interface with DE2 FPGA. The Hootoo U19-A Night Vision Webcam (12.0MP) was first purchased as a low cost, proof-of-concept video camera that could be bought in bulk at a cost of under $3.00 and which could be experimented upon with low risk.

### FPGA

Altera DE2 Development and Education was planned to be employed in the project to perform computational tasks in image acquisition, processing and gaze calculation. The board features an Altera Cyclone II 2C35 FPGA and can be easily interfaced with Terasic D5M camera, personal computer and video graphics array (VGA) monitor.

## Part II: Hardware Automation and Online Data Collection

After initial tests were completed using the first hardware setup, it was determined that IR illumination would be used to meet the design goals. The headrest and monitor were left unchanged. However, the red LEDs were replaced with IR LEDs, and IR LEDs were added to provide coaxial illumination to the camera lens. The camera was updated several times. The illumination source control microprocessor and circuitry were added. A fixed location for the camera was included. A new power source was used that would allow power to be used from any wall socket. Figure VII shows the final hardware setup. Each component will be described below.



The physical hardware configuration.

**1:** The Arduino controller, current sources, control FETs, and other circuitry.

**2:** The camera and coaxial IR LED illumination source.

**3:** The glint IR LED illumination sources.

**4:** The power source.

FIGURE VII
Final Hardware Setup

### BLUE BOX CIRCUITRY

The automation circuitry was contained in the blue project box seen in Figure VIII. Inputs consisted of 12-20V DC power (Port 1), coaxial LED power and camera V-sync signal (Port 2), Glint LED power (Port 3), and Arduino programming USB access (Port 4). The electronics consisted of an Arduino Duemilanova, a 311 comparator, an INA121P differential amplifier, two IRF740 nMOS FETs, and two 80mA tunable current sources built from LM317 voltage regulators (Figure IX). The connector diagram can be found in the Appendix.

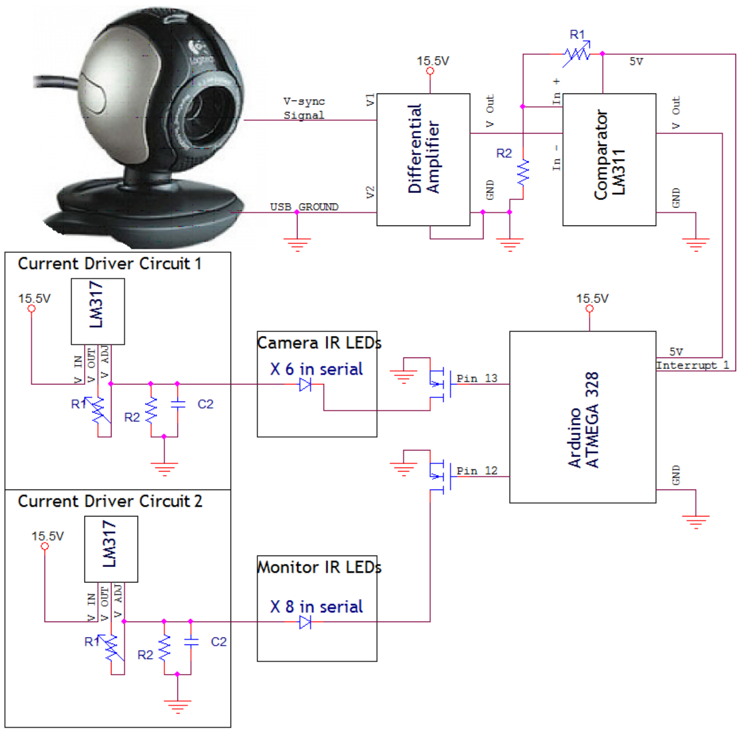


FIGURE IX
Schematic of Blue Box circuitry.

## *INA121P Differential Amplifier and 311 Comparator*

This integrated circuit was included to handle the floating voltage level of the USB input for the camera. Since the USB ground is not equivalent to earth ground, the differential amplifier was required to measure the V-sync signal relative to the common ground of the remainder of the system. In order for the system to work, the Arduino can only be programmed while the remainder of the system is off. This circumvented the Arduino power problem with minimal additional circuitry. The differential signal was input into the comparator. The comparator output a digital signal that was HIGH when the V-sync pulse was HIGH and LOW when the V-sync pulse was LOW. This signal was read by an Arduino as an interrupt signal.

## *IRF740 Power nMOS FETs*

A power nFET was used as a switch to toggle the current supply on and off for each LED set (the coaxial and the glint sources). This allowed fast switching on the magnitude of several nanoseconds that easily allowed a frequency, and thus a frame rate, of 30FPS to be achieved. The FETs were equipped with heat sinks to allow high current (rated at up to 10A). Actual currents were between 75-85mA.

## *Arduino Duemilanova*

An Arduino Duemilanova microcontroller was used to switch the nFETs on and off at the appropriate time during a data acquisition cycle. The V-sync was HIGH when an image was being captured. The Arduino read this signal as an interrupt and toggled both FETs, which were always in the opposite state of one another. Due to the periodic nature of the data collection and the fact that the V-sync signal was output at every image capture, the IR LED illumination source was guaranteed to switch between each image capture.

## *Current Sources*

A current source made from an LM317 voltage regulator was constructed for each LED illumination source. The current supplies were made to be tunable so that the brightness of each LED source could be equated. Since there were six coaxial LEDs and eight glint LEDs, slightly more current was required by the glint source LEDs than the coaxial source LEDs. The currents were adjusted so that the illumination was roughly equivalent for both sources. This resulted in a very well extracted pupil center when the images were subtracted (*see Algorithms*). The current for the coaxial LEDs was 75mA. The current for the glint LEDs was 85mA.

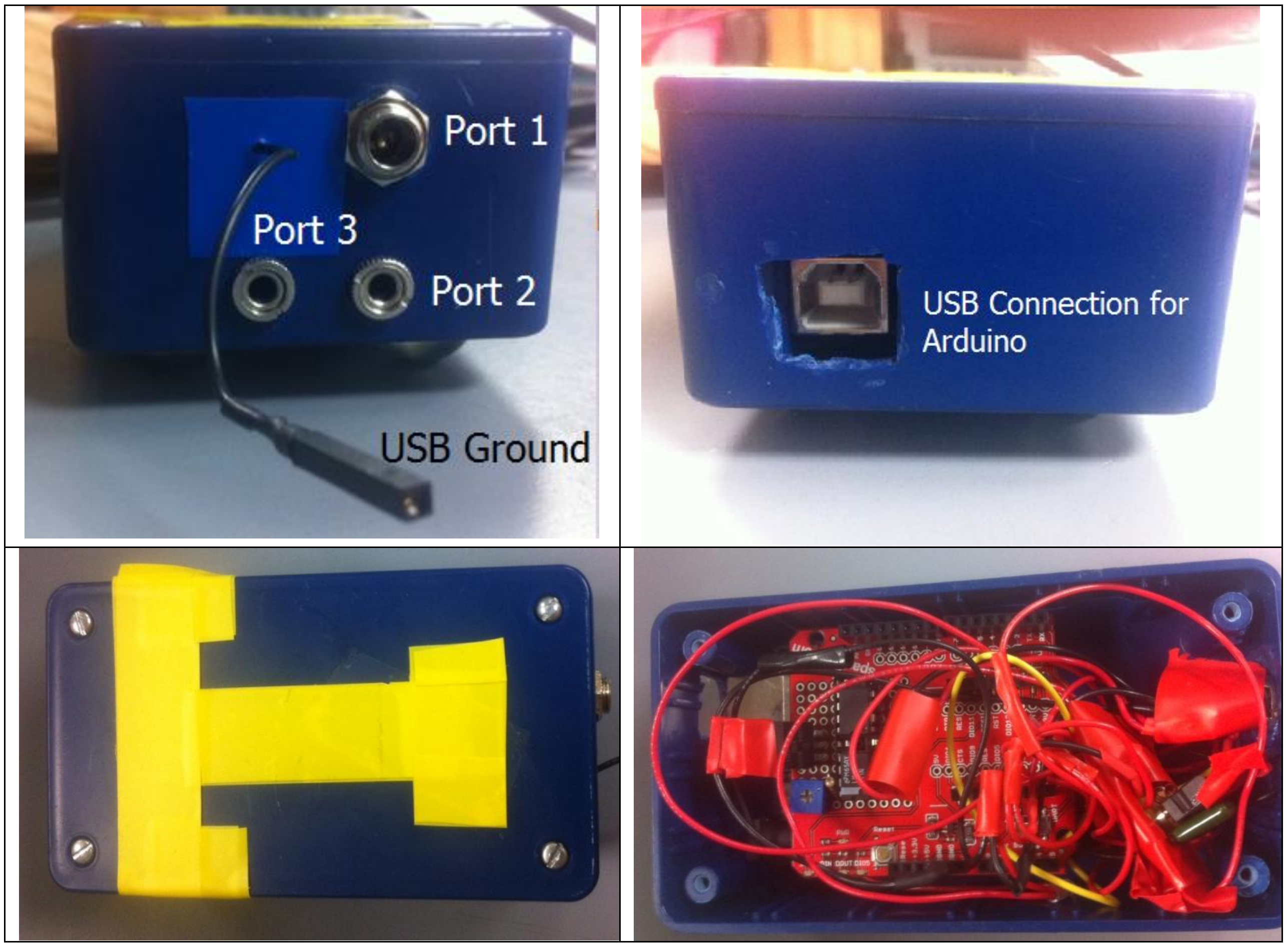


FIGURE X
Blue Box.

## LEDs and Monitor

The monitor was modified with a clamp to hold the camera at desired angles. The coaxial LEDs were mounted in the monitor frame to ensure that the location was static during development and for aesthetic value. The IR LEDs used were 875nm wavelength with a 60° output angle and 100mA maximum current.

## Camera and Lens

A Logitech Communicate MP camera was used to capture images of the user's eye. The 1.3 MPx camera did not have a zoom lens attached to it. Therefore, a lens holder and 25mm M12 lens were added to the camera. Initially, a lens with a larger diameter was used. The width of the lens did not allow LEDs to be brought closer to the image sensor. The further the LEDs were from the camera image sensor, the less pronounced the bright pupil effect was for a given distance between the camera and the user. As a result, a lens with a smaller diameter was used. The hot glass was replaced with an IR filter to make the camera sensitive to only IR light. Similarly, to automate the process of toggling power between two different sets of LEDs (glint source and bright pupil source), the V-sync signal was read through a wire drawn out of the V-sync node in the camera PCB. Figure XI shows the camera PCBs taken out of their casing. The larger PCB is connected to the computer through the USB cable. The smaller PCB holds the image sensor and the camera lens. Figure XII shows the V-sync node in the smaller PCB board. The camera could operate at 5, 10, 15, 20, and 30 FPS. The FPS limit was determined by this hardware, as image processing speeds were fast enough to allow the maximum frame rate of 30FPS to be used for image capture (*see Algorithms: Final Software*).

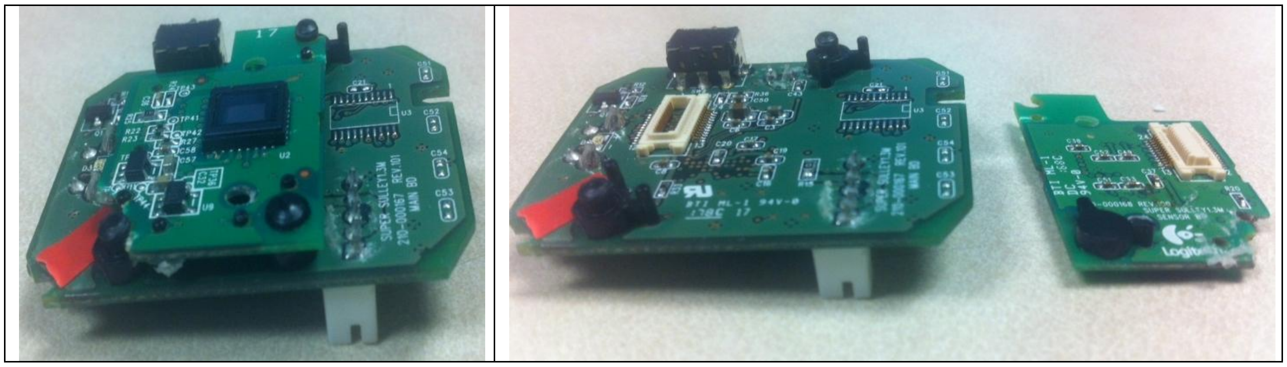


FIGURE XI
Camera PCBs out of the casing.

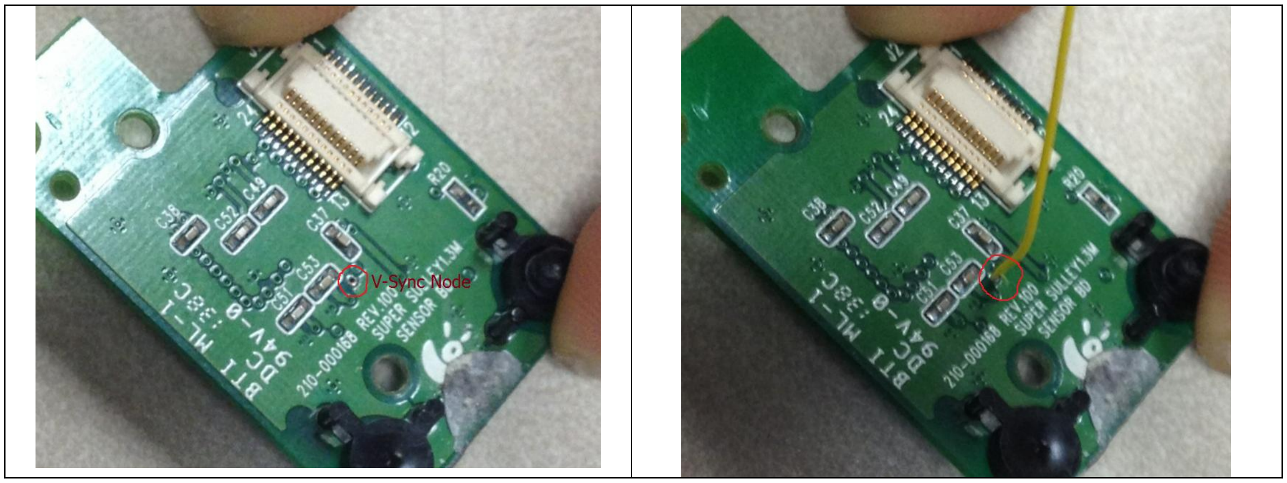


FIGURE XII
Connecting the wire to the V-sync node on the camera PCB.

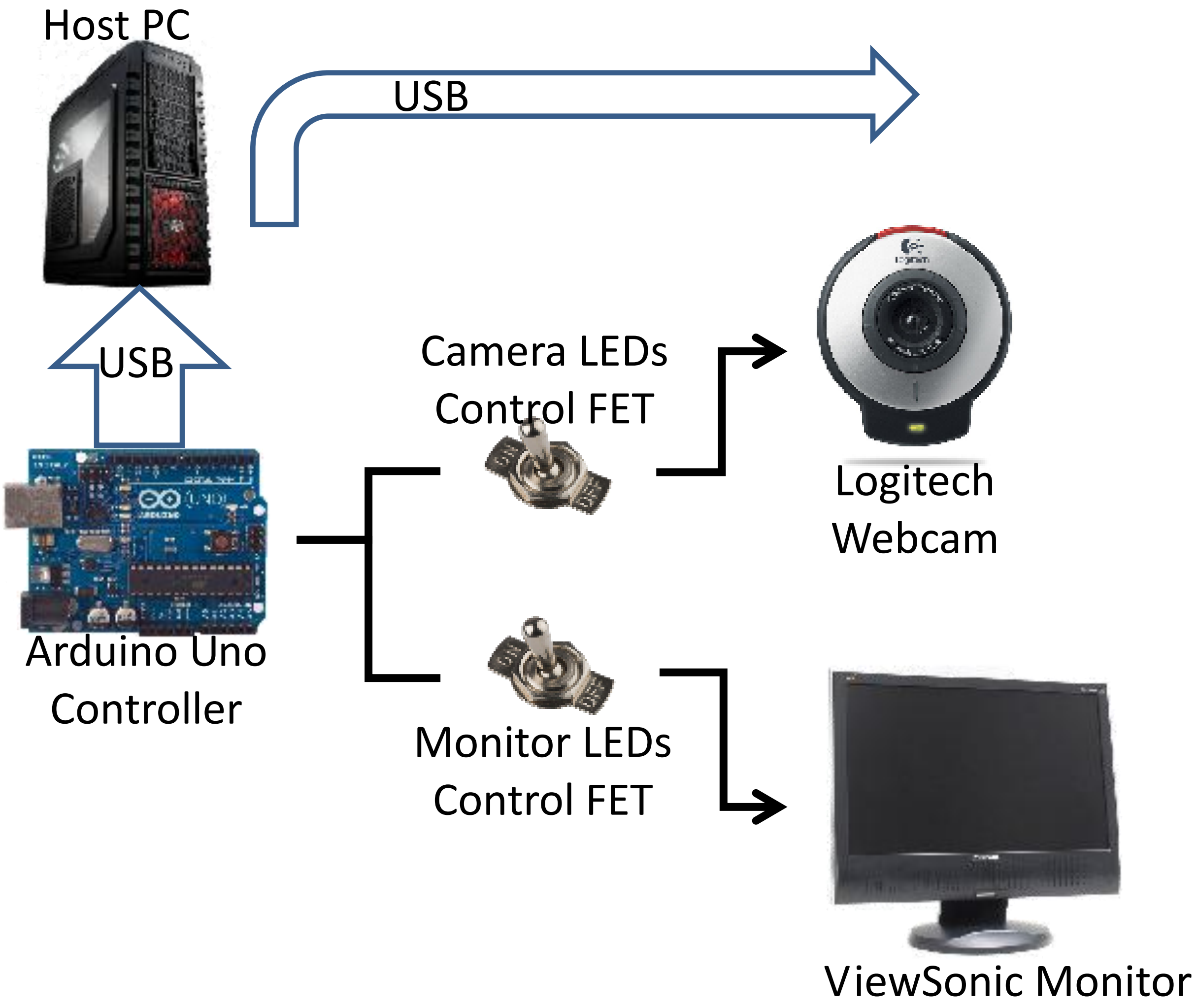


FIGURE XIII
Final Hardware system architecture.

## VI. ALGORITHMS

### PART I: VISIBLE LIGHT APPROACH

### INITIAL DATA ACQUISITION AND ANALYSIS

The initial data acquired was analyzed to find which features could be used to determine where the user was looking on the screen. An image was captured with the user looking at each grid location. This was done with the Cannon digital camera. The prominent features in the image were the location of the pupil and the reflected screen of the monitor. While the monitor itself was considered as a glint source, the reflection was found to be too dim to be a reliable indicator of the screen location. However, it was noted that the center of the pupil could reliably point to where on the screen the eye was looking. When the user looked at the corner of the screen, the pupil center was on the corner of the screen reflected on the cornea. Thus, the cross-ratio method was verified as a plausible route to pursue.

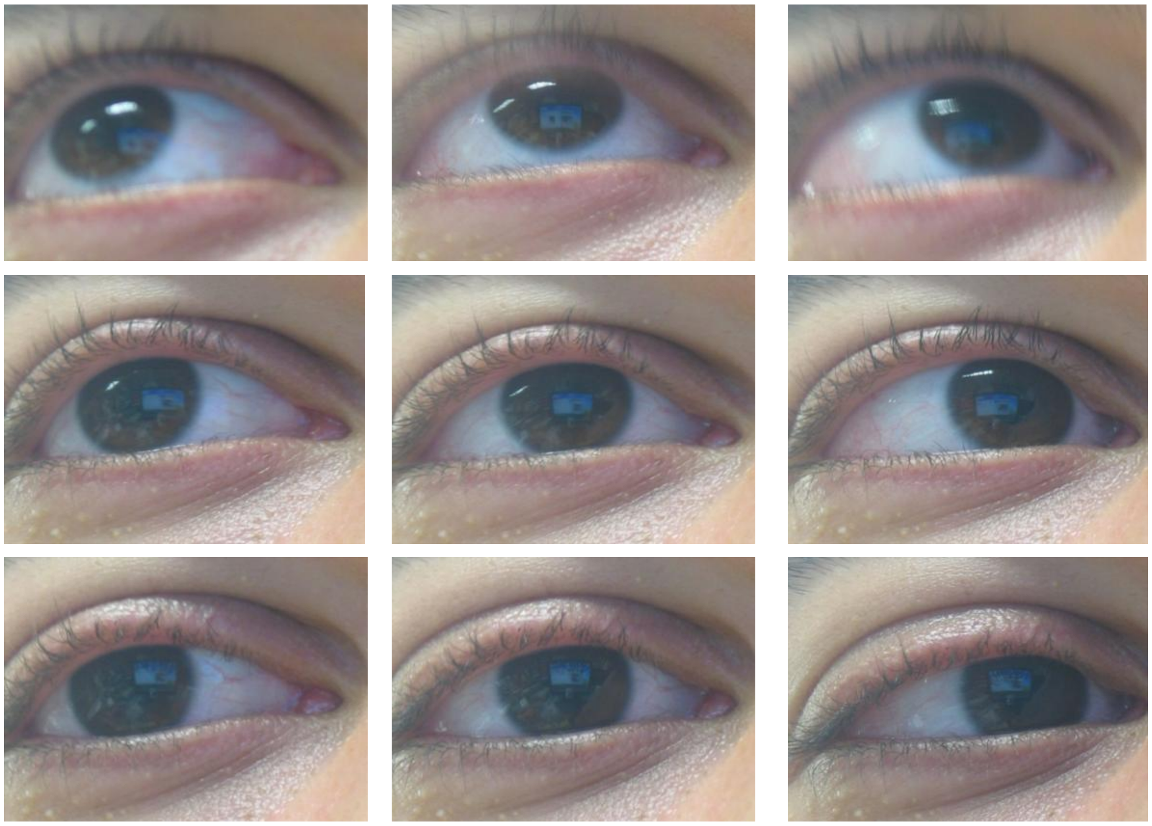


FIGURE XIII

Sample data for 3x3 grid.

### SATURATION AND VALUE THRESHOLDING FOR PUPIL DETECTION

This method is based on the observation that while using the camera flash during image acquisition, a user's pupil is the darkest region observed. More specifically, the pupil appears black because most of the light in the visual spectrum that enters the pupil is absorbed by the retinal tissues inside the eye. This property of the retina and pupil can allow for pupil center detection. The darkness of the pupil center is observed in the grayscale conversion of the raw image. However, other dark areas are also be detected. This is overcome by analyzing the saturation plane of the image (the pupil is a very saturated region of the eye image). To achieve this, a Hue Saturation and Value (HSV) conversion of the red-green-blue (RGB) image obtained from the camera is used. Figure VII shows the color space conversion from RGB to the cylindrical coordinate system, HSV. Hue is a measure of displacement from a neutral color (i.e. the red, green and blue intensities are all equal). This spectrum ranges from 0˚ (red) to 120˚ (green) and 240˚ (blue). In addition to this component, this color space is described by value and saturation. More specifically, the value of an image is its brightness, or its grayscale conversion. Lastly, saturation is a measure of color density, or more precisely, the ratio of color or chroma to the value or brightness.

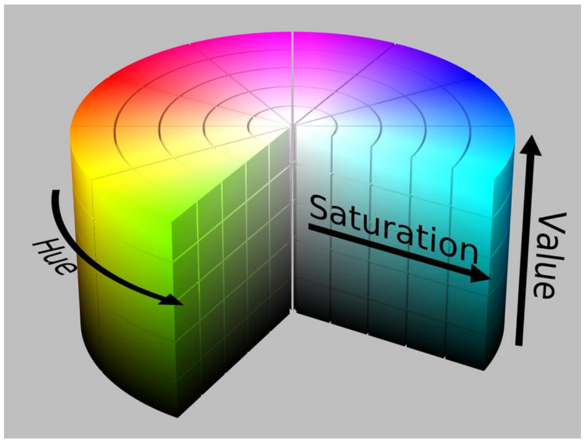


FIGURE XIV
HSV Color Space

This thresholding algorithm can be used with the data collection hardware setup described earlier for the overall project. The LEDs at the four corners of computer screen are not necessary, but the camera flash must be enabled. A dynamic threshold is applied and the pupil location is extracted.

For this algorithm, the MATLAB image processing library was used. The image processing algorithm can be divided into following parts:

a. Image loading.
b. HSV conversion and plane extraction.
c. Threshold value calculation.
d. Image filtering
e. Center calculation

a. Image loading:

```
clear all;
filename = uigetfile('*.jpg', 'Pick a
imagefile');
picmtrx = imread(filename);
image = imresize(picmtrx,.25);
```

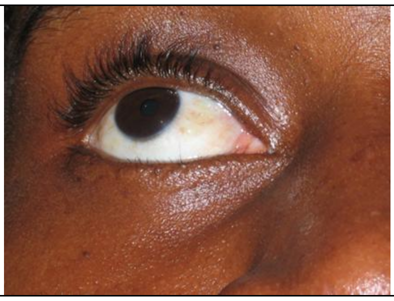


Figure XV: Image loading MATLAB code and sample image

The image obtained from the camera is a JPEG image. The program reads the image and stores it in a variable called picmtrx. Because the image is larger than necessary, it is reduced to a quarter of its original size. Resizing also allows for faster processing.

b. HSV conversion and plane extraction:

```
imagehsv = rgb2hsv(image);
imagehue = imagehsv( :,:,1);
imagesat = imagehsv( :,:,2);
imageval = imagehsv( :,:,3);
imageval1 = imageval;
imageval = im2uint8(imageval);
imagesat = im2uint8(imagesat);
```

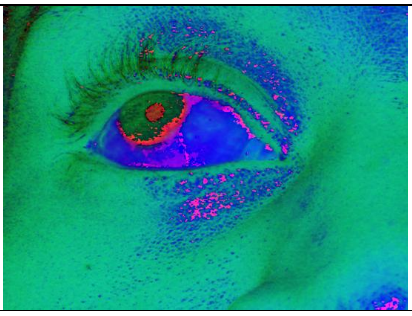


Figure XVI: HSV Conversion of Raw Image

The image is converted to HSV format and Saturation and Value planes are extracted. Due to the absorption of visible light by the retinal tissues, the pupil is one of the darkest parts in both Value plane and Saturation plane. Figure X displays both the value and the saturation planes of the HSV space of the image. Also, both the planes are converted to 8bit format for faster processing.

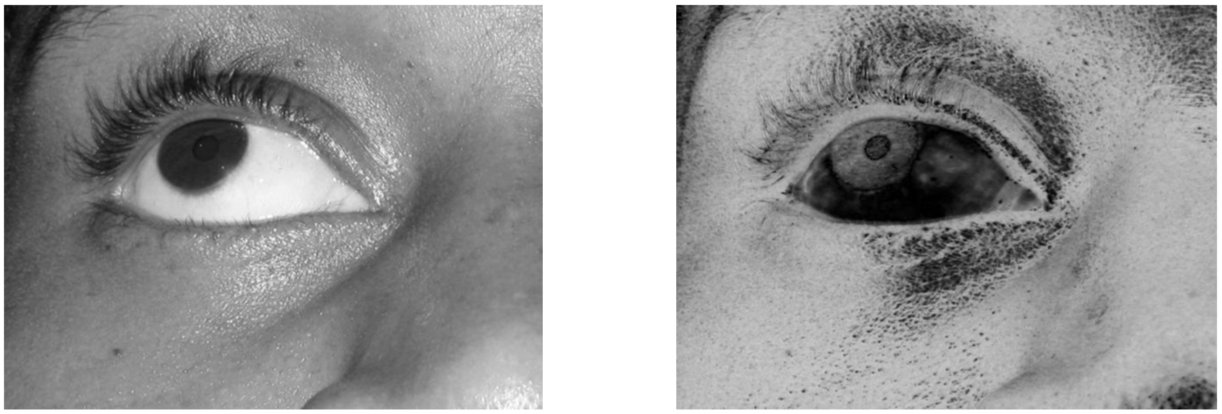
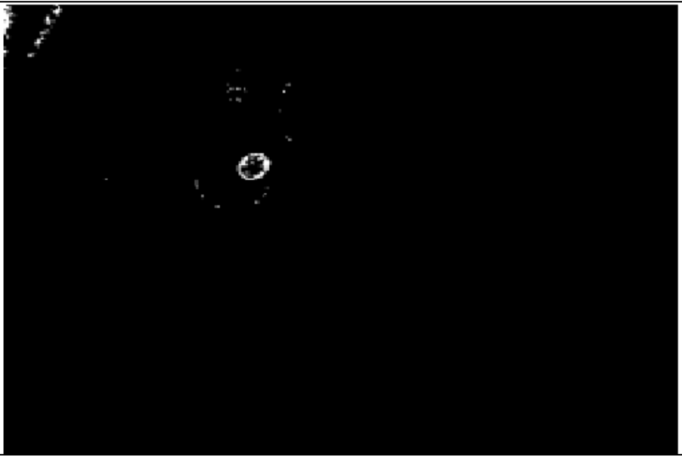


FIGURE XVII

C Value and Saturation Planes of HSV Representation

c. Threshold value calculation:

|  |  |
|---|---|
| `imagehist1 = imhist(imageval); imagehist2 = imhist(imagesat); j= 1; sum = 0; tenthousandtage = 0; while(tenthousandtage<400) sum = sum + imagehist2(j); j = j+1; tenthousandtage = (sum/(426*568))*10000; end` |  |

```
imagehist1 =
imhist(imageval);
imagehist2 =
imhist(imagesat);
j= 1;
sum = 0;
tenthousandtage = 0;
while(tenthousandtage<400)
        sum = sum +
imagehist2(j);
        j = j+1;
        tenthousandtage =
(sum/(426*568))*10000;
end
```

Figure XVII: 4th Percentile of Saturation **AND** Value Planes

The pupil is one of the darkest regions in both the value and saturation planes. An empirically determined threshold value of 4th percentile darkest pixels in both planes is utilized. Later anything over the limit is converted into 8-bit black (0) and anything under the limit is converted into white (255). Finally, any pixel that is not white (255) in both the planes is converted to black (0).

d. Image filtering:

```
bw = imfill(bw, 'holes');
bw = imclearborder(bw);
bw = medfilt2(bw,[20 20]);
```

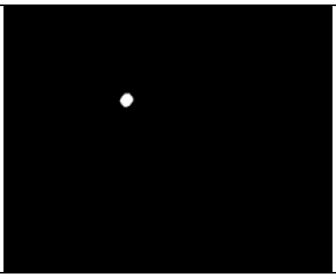


Figure XVIII: Filtered pupil detection

The image obtained after just thresholding does not provide an ideal signature of the pupil, as seen in Figure XI. Instead, there are small specks and dark areas that do not correspond to the pupil. A solid signature of pupil is obtained through the function imfill, which fills any holes. Then noise pixels are removed using a median filter. The median filter removes anything that is less than 20x20 pixels in size. The obtained image from the filtration accurately separates out the pupil from rest of the image, as shown in Figure XII.

e. Center calculation

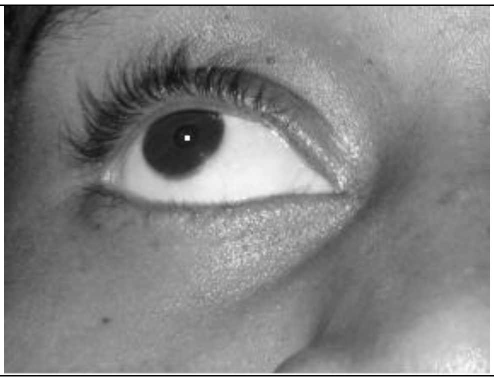
| | |
|---|---|
| `x = round((maxx+minx)/2)`<br>`y = round((maxy+miny)/2)`<br>`imageval(y,x)=255;`<br>`figure, imshow(imageval);` |  |

Figure XIX: Pupil center superimposed on value plane

A simple algorithm that takes the midpoints between the minimum and maximum x and y values of the pupil region is used to pin point the pupil center. The end result is shown in Figure XIII.

*<u>Analysis</u>*

The algorithm was very effective in finding pupil center when camera flash was used. The code was tested with people with different eye shapes looking at different parts in the screen. Among the 45 images obtained from 5 people, the code was successful in estimating pupil center 89% of the times. Figure XIV shows four different subjects and the pupil center estimation using the described algorithm.

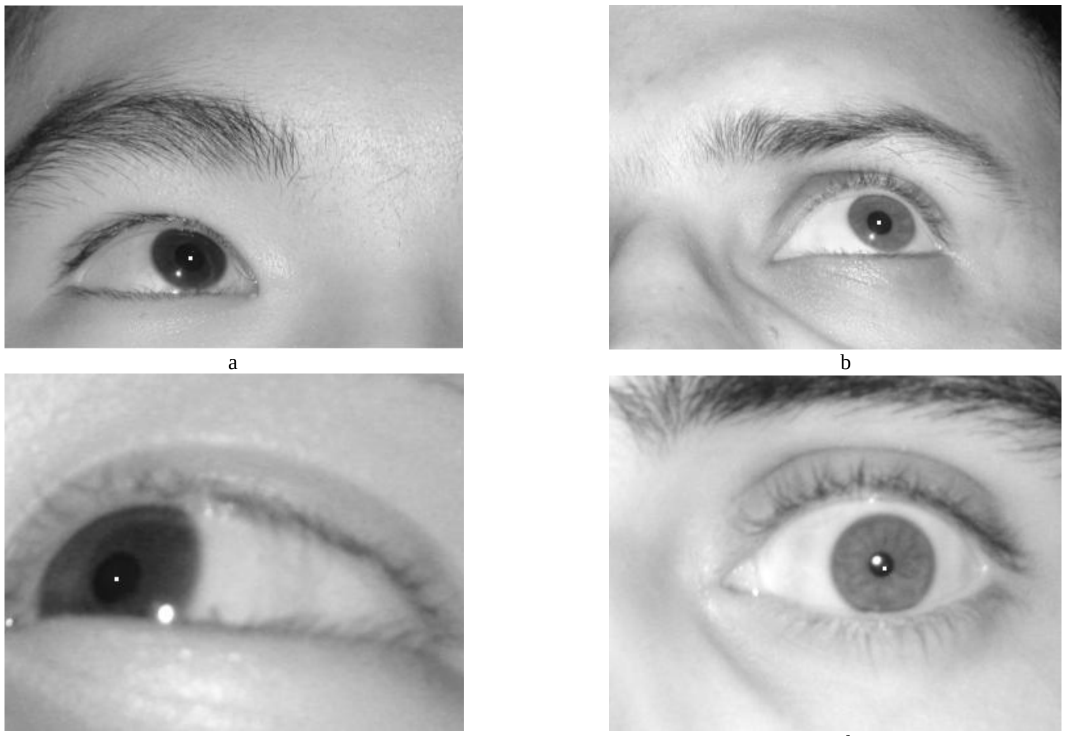


FIGURE XX
Pupil Center Estimation of Subjects a, b, c, d

The algorithm was then tested with images acquired without using the camera flash. The result in this case was quite different; the algorithm was not successful. Because of the corneal reflection of ambient light (which would otherwise be saturated with camera flash), the pupil did not appear as the darkest region in the images. Instead, reflections on the pupil created very bright areas on the pupil. As a result the algorithm was not able to determine a reliable estimation of pupil center. Figure XV displays the original image with corneal reflection and the result of the thresholding algorithm. Finally, a similar algorithm to this was selected for the IR illumination approach. Due to the lack of ambient noise, the detection of the pupil center was much more effective in a natural (no-flash) environment (*see Algorithms: Part II*).

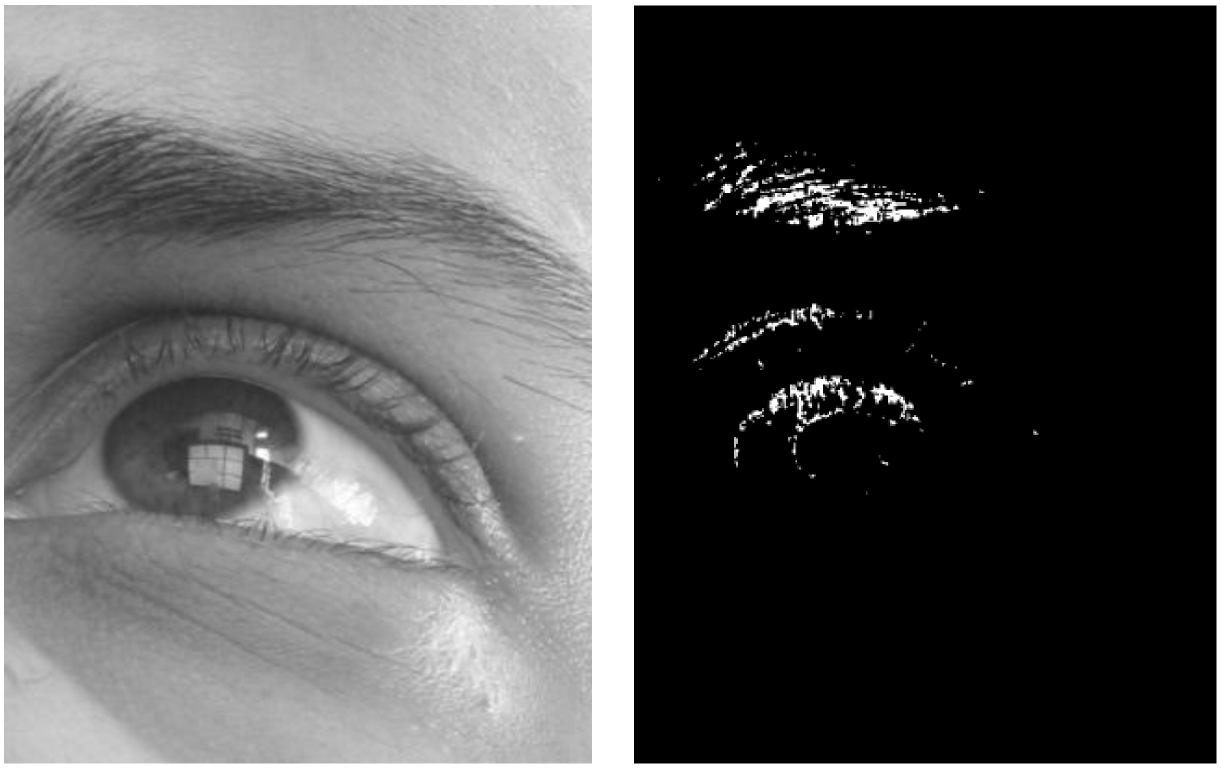


FIGURE XXI
Algorithm Failure Without using Camera Flash Illumination

## GLINT LOCATION

As stated above, an environmental signal is introduced to the system that provides a common reference point for each captured image (see *Theory*). In this case, this signal creates a glint or reflection on the eye of the user. For the cross-ratio method, these glints must be positioned at the corners of the screen. This projects a polygon onto the user's eye which may be used as a reference location for the screen when creating a correlation matrix. The glints created on the eye must be extracted by the image processing or the information cannot be used for calculations.

## *Common Mode Rejection Method*

Common mode rejection (CMR) is a signal processing technique that extracts the desired signal from a noisy system. When the system has a low signal to noise ratio (SNR) it is difficult to detect the signal itself or small changes in the signal value. If the noise is constant, it can be filtered by first measuring the output of the system without an applied signal and subsequently removing any common aspects when measuring again with the signal applied. In electronics, this occurs when a signal represented by small voltage fluctuations is superimposed on a large voltage offset. In both instances, the voltage offset is present: hence, it can be detected before the signal is applied and removed. In this way, the commonalities of the two outputs are rejected.

For the Gaze Controlled Human-Computer Interface, this approach aims to reject any ambient light common to the system for both an image taken when the glint source is inactive and an image taken when the glint source is activated. A simple and effective way to accomplish this is to take the absolute difference between the two images. The result highlights any changes between the two images and rejects any commonalities between them. If the frame rate of the camera is sufficient for the user to not move between image captures, the images will be identical except for the applied signal, which will be highlighted.

## *Visible Light Glint Source*

First, the common mode rejection method was attempted with visible light glint sources (650nm red LEDs). A test LED was configured next to the camera so that the intensity of the reflection would be maximized. Two images were taken in rapid succession. One image had the LED glint source activated and the other had no glint signal activated (*see Figure XII below*).

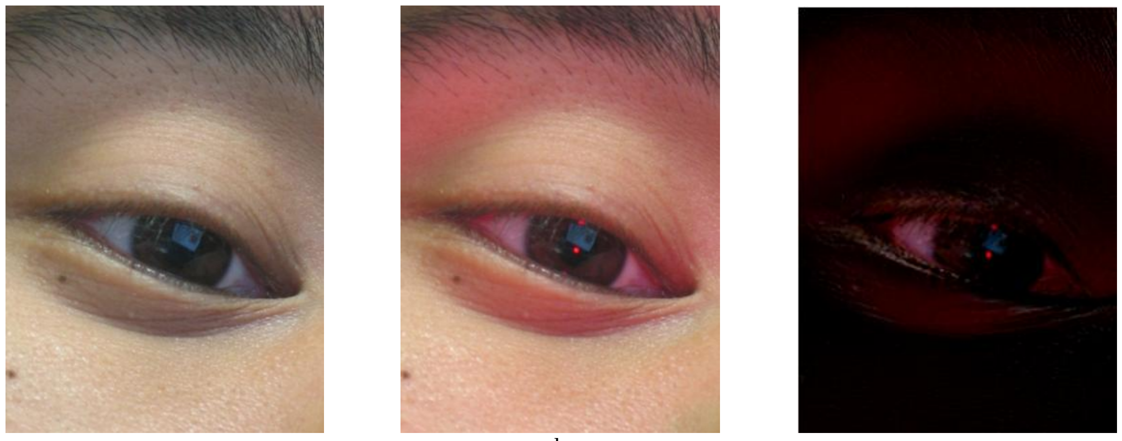


FIGURE XXII
*a)*Raw image without glint source illumination. *b)* Raw image with glint source illumination.
*c)* Common mode rejection, subtracted a) from b).

This approach was found to be moderately effective for highlighting the glint locations in the image. The glints from Figure XVI were found by red plane extraction and thresholding (see Figure XVIIa). The ambient light created by the LEDs contributed to a substantial differential in the environmental conditions between the images. The sclera was thus highlighted using this method because it reflected the red glow of the LEDs. From the result it is difficult to extract the

LED glints. However, this method yielded far better results than extraction of the LED glint from the raw image (see Figure XVIIb). The LED was not bright enough to be detectable from the raw image alone. Brightening the LED was considered a potential solution to this problem but may cause discomfort to the user. A similar approach was used with IR illumination and the results were much improved due to the lack of ambient noise (*see Algorithms: Part II*).

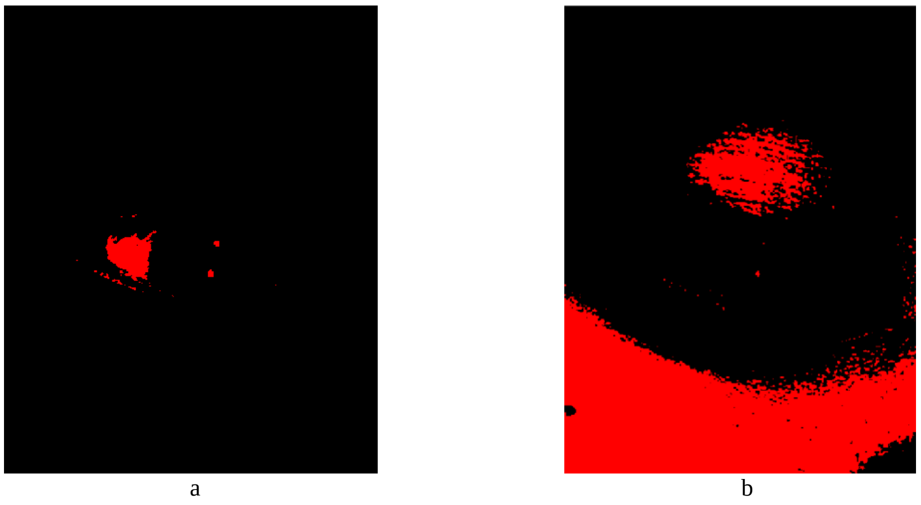


FIGURE XXIII
*a)*Output of CMR technique, glint detection. *b)* Glint detection from raw glint illuminated image.

## PART II: VISIBLE LIGHT APPROACH INFRARED ILLUMINATION TECHNIQUE

The algorithms developed for the glint and pupil detection described above were only moderately successful. They were not found to be robust enough to be implemented in a user-friendly design. One of the widely used methods for gaze detection was found to be bright pupil/ dark pupil-based CMR. As the human eye acts as a retro reflector, it is possible to obtain a bright image of pupil when the user's eye is illuminated with IR source almost coaxial to the camera lens (*see Theory*). However, when the IR illumination is not coaxial to the camera lens, the bright pupil effect does not occur. Thus a system that can take images of both the coaxial IR source illuminated eye and the non-coaxial IR illuminated eye can be constructed, and CMR can separate out the pupil from the rest of the image. This approach used the IR hardware setup (*see Hardware, Part II*).

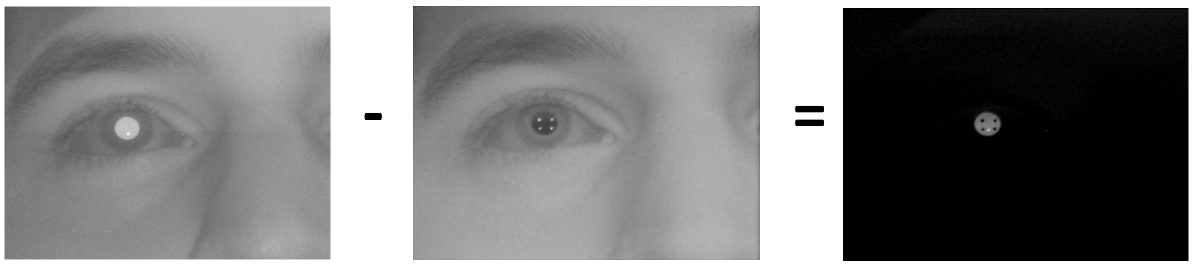


FIGURE XXIV
Example of final data provided by updated hardware setup. The dark pupil image is subtracted from the bright pupil image to remove ambient noise. Only the actual signal, the pupil and glints, remains.

Although the method is conceptually simple, there are some design requirements that are necessary to make this method useful for the project's purpose, which are as following:
1. Illumination from the IR emitters should be enough to properly illuminate the eyes.
2. Illumination from the coaxial and non-coaxial emitters should be approximately equal to one another.
3. The camera should be sensitive to IR light.
4. The frame rate of the camera should be fast enough make the CMR method work.
5. The camera's image acquisition and the toggle between the two types of illuminator should be synchronized properly.

These goals were met with the updated hardware model (*see Hardware, Part II*). A sample of the actual data collected can be seen in Figure XXIV. Adjustments were made to the current though each illumination source so that the CMR image (far right) would have no ambient noise in it.

*Software*

Once the final prototype hardware was completed, a method of controlling image acquisition, calibrating the system for the user and executing our classification algorithm was necessary. Moreover, a user-interface that provides gaze estimation feedback was required. To that end, Python 2.7 was the programming language selected (for its versatility, image handling as well as graphical user interface development). The program functions can be divided into three main subgroups:

1. Image Capture and Gaze Classification
2. Calibration
3. Graphical User Interface

First, we will import the necessary libraries and declare global variables. The most utilized package for the image acquisition and classification portion of the program is the OpenCV library. This package allows for polling the Logitech camera for an image as well as several image processing techniques. Next, the NumPy library was imported to handle matrix operations and basic array manipulation. For the graphical user interface, Tkinter provided the means to draw shapes as well as text on the entire monitor screen. In addition to this, the VideoCapture library was used to manipulate webcam settings.

The following code imports these libraries:

```
import sys
sys.path.append("C:\OpenCV2.2\Python2.7\Lib\site-packages")
import cv
import numpy as np
import time, string
import copy
import math
from Tkinter import *
from math import*
import time
from VideoCapture import Device
```

## *Image Capture and Gaze Classification*

While the program is running, images are constantly polled from the webcam images are subtracted from each other in pairs of two to obtain the bright pupil image. The image acquisition is performed at a rate of 30 fps. To determine which of two consecutive images is the bright pupil and which is the dark pupil, a simple static threshold is used (the bright pupil image is always brighter than the dark pupil. We subtract one from the other, and then vice versa. The brighter image should be the subtraction of the dark pupil from the bright pupil). The dark pupil image is then subtracted from the bright, as shown in Figure XXIV. The pupil center is continuously determined via a static threshold, and a 19x19 pixel median filter used to smooth out the pupil boundary as well as reject small bright spots.

Once the pupil coordinates are determined, a small subarray of the entire dark pupil image is extracted, particularly the small area surrounding the calculated pupil center. We know that the glints will only be within a certain distance of thee pupil center. This is used to achieve two things:

1. Reduce computational requirements
2. Reduce effects of other bright spots, reject unwanted signals

This captured result is shown below:

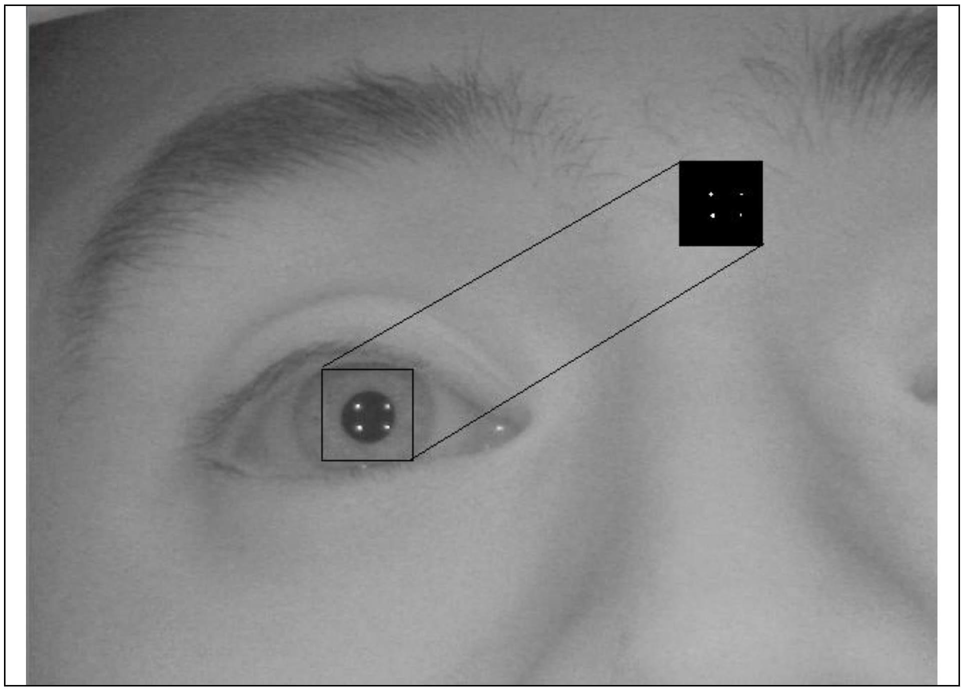


Figure XXV: Cropping and thresholding the area around pupil center to find glints

This image is then split into quadrants in which glint reflections are searched for, again via a static threshold. If the cross ratio method is ideal, then the pupil center will never leave the polygon formed by connecting the four corneal glints so long as the subject constrains their gaze to the monitor. If this assumption is met, then there will be exactly one glint in each quadrant:

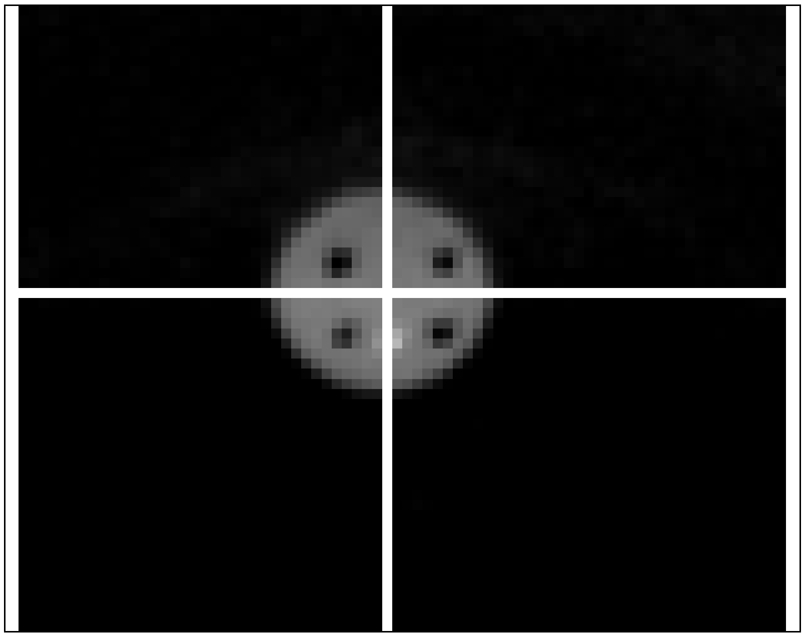


Figure XXVI :Separation of 4 glints from the cropped image.

Based on the calculated distances of the glints to the pupil center, a point of gaze is estimated. To estimate the point of gaze, the relative distances of the glints and pupils are projected onto the monitor via a linear model for both horizontal and vertical directions, as shown in the following snippet of code:

```
Xr = (XH/(XL+XH))
Yr = (YH/(YL+YH))
#print(Xr, Yr)

Yr = ((Yr-0.2)*(1/(.8-.2)))
Xr = ((Xr-0.23)*(1/(.7-.23)))
```

In this case, Xr and Yr are the calculated ratios of the distance of the pupil vertically and horizontally to the glint locations respectively. Then the values are transformed via two linear models that relied on empirical testing for initial calibrated settings. These linear models are created via a simple point-slope methodology.

It should be noted that it is these equations whose coefficients will be modified after the calibration phase of the software.

Once the gaze location is determined a small orange dot is placed on the monitor in the Tkinter canvas. The size of the orange dot correspond to the ‘confidence’ of the gaze estimation. To achieve this, a buffer of ten measurements is created, and the sparseness or distribution of data within the buffer determines the ‘confidence’. Moreover, this buffer also takes an average value to smooth the gaze estimation (recall that Xr, Yr store the glint estimation coordinates). After each iteration of the while loop, the buffer is reloaded in a FIFO fashion:

```
            bufx1 = Xr
            bufy1 = Yr
            Xr =
(bufx1+bufx2+bufx3+bufx4+bufx5+bufx6+bufx7+bufx8+bufx9+bufx10)/10.0
            Yr =
(bufy1+bufy2+bufy3+bufy4+bufy5+bufy6+bufy7+bufy8+bufy9+bufy10)/10.0

            bufy2 = bufy1
            bufy3 = bufy2
            bufy4 = bufy3
            bufy5 = bufy4
            bufy6 = bufy5
            bufy7 = bufy6
            bufy8 = bufy7
            bufy9 = bufy8
            bufy10 = bufy9

            bufx2 = bufx1
            bufx3 = bufx2
            bufx4 = bufx3
            bufx5 = bufx4
            bufx6 = bufx5
            bufx7 = bufx6
            bufx8 = bufx7
            bufx9 = bufx8
            bufx10 = bufx9
```

The above described methodology is continuously iterated through a while loop, whether in the calibration phase or not.

*Calibration/Graphical User Interface*

Throughout both the calibration phase and the tracking phase, a Tkinter canvas is used to construct, destroy and reconstruct images. To calibrate, firs ta canvas is constructed and key bindings are assigned:

```
self.canvas = Canvas(master, width = w, height = h)
        self.text = self.canvas.create_text(w/2,h/2, text="Look at calibration
point and press return 3x.", fill="black", font=("Helvectica", str(20)))
        self.r=self.canvas.create_oval(0, 0, 1, 1, fill="cyan")
        self.circ = self.canvas.create_oval((4*w/5)-(rad), (h/5)-(rad),
(4*w/5)+(rad),(h/5)+(rad),fill="red")
        self.canvas.pack()
```

```
#Bindings
self.tk.bind("<Escape>", self.end)
self.tk.bind("<Key-Return>", self.increase)
self.tk.bind("<BackSpace>", self.kill)
```

In the self.increase method, the calibration portion of the code is executed. In particular, six red dots are placed sequentially on the monitor at specific locations (1/5, 4/5 of the screen vertically, and 1/5, 1/2, 4/5 of the screen horizontally). When the ‘Return’ key his pressed, the calculated gaze at that instance is recorded (based on initial settings). For each calibration point, three ‘Return’ key strikes are used to record three gaze estimations. Two conditions result in recalibration for that point:

1. The estimated gaze is points are two spread apart
2. The average of the estimated gaze point is egregiously off from expected gaze

Figure XXVII shows the Tkinter calibration screen:

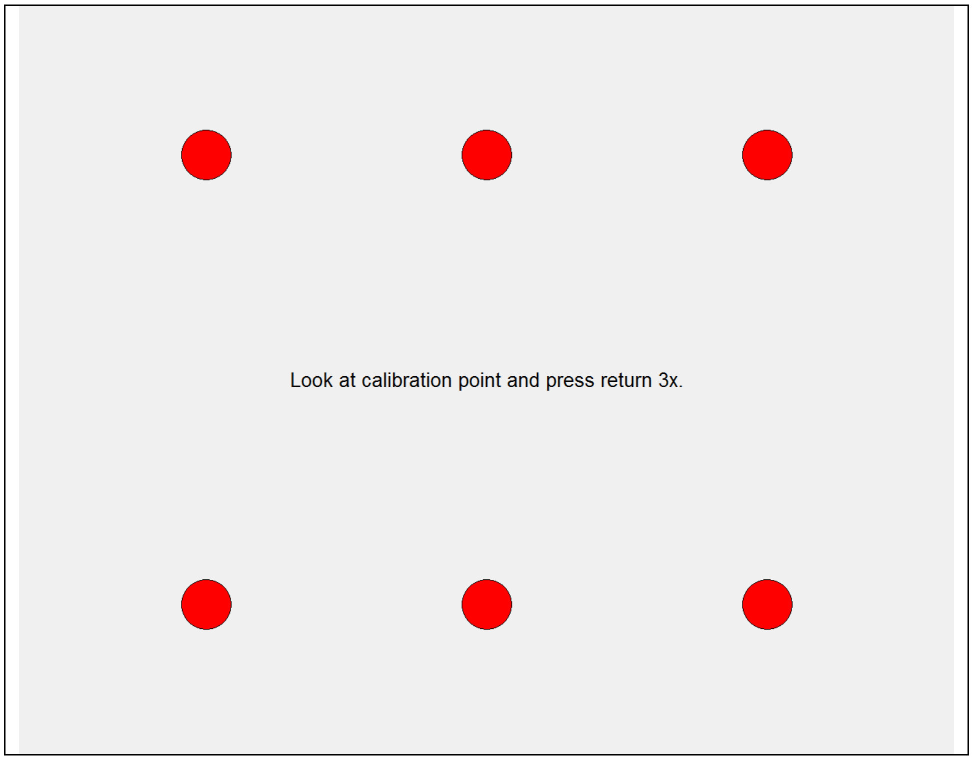

Look at calibration point and press return 3x.

Figure XXVII: Calibration Screen

During this time, the gaze estimation algorithm is continuously running to give the user some sort of feedback as to how far away their gaze estimation is relative to the calibration points. Once the calibration is complete, the linear models are adjusted to the calibration data and the calibration text and dots are removed.

For demonstration purposes, the orange dot (corresponding to gaze location and confidence) was programmed not to refresh. The results of a typical calibration and user thereafter are shown below in Figure XXVIII:

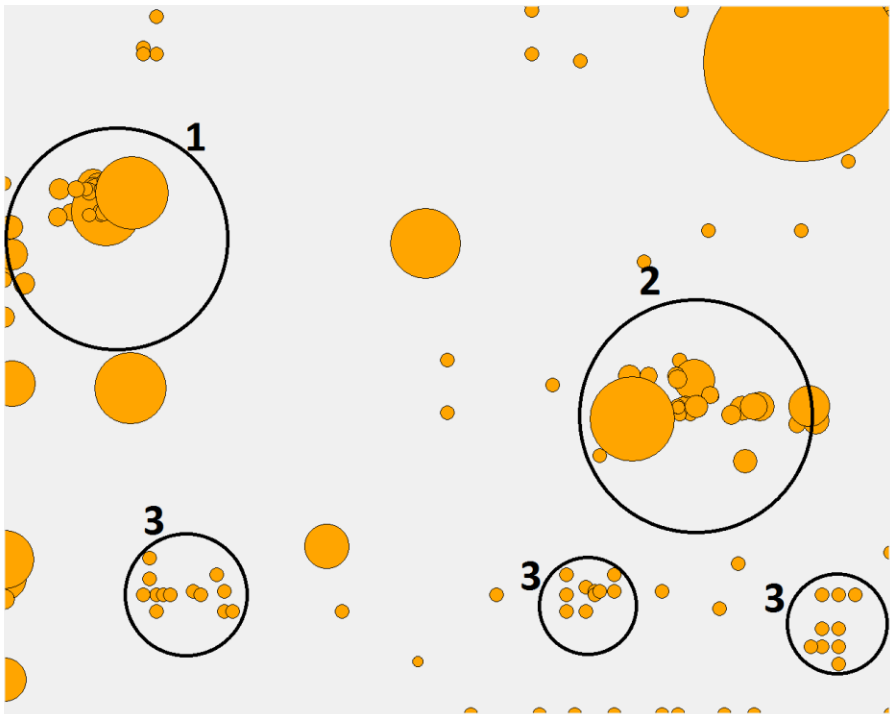


FIGURE XXVIII
Time lapse of user interface. 1: The user looking at a location on the left side of the screen. 2: The user looking to the center-right. 3: Locations acquired during calibration (fixed size).

As shown, we can see clear clustering of estimated gaze. Furthermore, we can see the user's initial gaze estimation when attempting to look at the bottom three calibration points (these gaze estimation are clearly off a bit). The groupings of these clusters are very compact, suggesting that the dot size would be relatively small. Compared to the dot in the upper right hand corner, the confidence levels at these grouped/clustered points are very small, corresponding to high confidence.

Recall that the larger size of the dots means a larger variance in the position buffer and a larger uncertainty of position. Thus, the isolated large dots are acquired during eye movement across the screen. The dots become smaller when the eye position is moving slowly or is stationary.

## VII: RESULTS AND GOALS MET

The goals set for this project at its commencement were the following:

1. Establish a non-intrusive environment for which POG is suitable
2. Detect corneal glints and pupil movements
3. Use the cross-ratio method to map eye movements onto the screen
4. Display Results in a Graphical User Interface
5. Create an embedded system that will allow real-time application
6. Expand system functionality through testing

All of these objectives were met. An automated, non-intrusive environment was established by which the user could operate the device without technical knowledge or discomfort. The user need only have his/her eye in the frame of the camera for the POG system to properly function. The camera lens is set to zoom at a particular distance, but this is adjustable. The headrest, which is included in the Hardware Setup, is not required but facilitates use of the system by guiding the user's head to the approximate position required. The hardware is compatible with any host PC and can be run from a wall outlet. The Blue Box circuitry is properly protected from unintended harm and is properly shielded to prevent failure. The final hardware has never failed to capture the desired dark/bright pupil images in succession.

Feature detection is fully automated. The user need not adjust his/her environment in any way (such as lighting adjustments) to ensure that proper feature detection and extraction takes place. The pupil detection is very reliable: in repetitive tests, the software never failed to detect the pupil center. Pupil detection only fails if there is more than one eye on the screen or objects that reflect IR light very well are introduced (such as spectacles). Contact lenses do not inhibit feature detection from properly working. The glint sources occasionally fail to be detected. This is the weakest link in the system. This occurs if a user is sitting too close to the screen, the camera is out of focus, or if the user is squinting excessively. When the glint source detection fails, the position detection becomes unreliable, and the system variance increases as a result. This is the main cause of flickering on the screen of detected position. If the glint detection is improved, the stability of the detected position will improve as well.

The image classification technique used is the cross-ratio technique (*see Algorithms, Part II*). This approach works very well in the center of the screen and leads to small deviations of less than 1cm. However, near the screen edges, if glint detection fails, the position can jump across the screen briefly before returning back to the desired position. This occurs most often near the bottom edge of the screen. The baseline resolution was a 3x3 grid. Instead, the system was made to be continuous.

The GUI constructed was designed to be user friendly and simple. When the calibration session terminates, the program automatically begins continuous, real-time display of the location of the user's gaze on the screen. Uncertainty in position was clearly shown by cursor size. The movement of the cursor was somewhat jittery. This was due to the 30FPS update. A larger position buffer in the software could smooth out the display.

The initial design of the system required an FPGA to ensure real-time functionality. The image processing code was designed with the desire to maximize throughput. The target was 30FPS, which was a hardware limit. This speed was met by avoiding methods employing quadratic time (such as nested for loops) and by cutting the image matrix to a 50x50 area of interest around the pupil center as soon as possible. Real time operation was achieved on the host PC and the excessive costs of an FPGA were avoided.

The system was tested on several students and faculty as well as all three principal investigators. Out of the roughly ten individuals who attempted to use the device, one failure was recorded. The reasons for this failure are unclear.

## VIII. COST

| Component | Quantity | Total. Price |
|---|---|---|
| Logitech Quick Cam MP<br>*http://www.amazon.com/Logitech-QuickCam-Communicate-MP-Black/dp/B0016JMS90/ref=sr_1_cc_1?s=aps&ie=UTF8&qid=1336283169&sr=1-1-catcorr* | 1 | $24.99 |
| Project Box<br>*http://www.amazon.com/Hammond-1591ASBK-ABS-Project-Black/dp/B0002BBQNM/ref=sr_1_1?ie=UTF8&qid=1336286127&sr=8-1* | 1 | $3.86 |
| Arduino Duemilanove Board<br>*http://www.amazon.com/Arduino-Duemilanove-Board/dp/B004A7L3NC/ref=sr_1_1?s=electronics&ie=UTF8&qid=1336286210&sr=1-1* | 1 | $19.95 |
| LED IrLED 875nm<br>*Jameco Part no. 1950622* | 14 | $5.46 |
| Instrumentation Op. Amp<br>*Digi-Key Part Number INA121P-ND* | 1 | $8.88 |
| LM317<br>*Digi-Key Part Number LM317TFS-ND* | 2 | $1.42 |
| IRF740<br>*Digi-Key Part Number IRF740PBF-ND* | 2 | $3.26 |
| LM311<br>*Jameco Part no. 23528* | 1 | $0.29 |
| Miscellaneous Components (wire, capacitors, resistors, connectors) | n/a | ~$10 |
| Total: | | $78.11 |

## IX. Future Work

This project was completed successfully. However, there are areas in which improvements can still be made. The cross-ratio method assumed a perfect polygon on the eye rather than one which is distorted by the curvature of the eye. Accounting for this could increase the precision of the calculated position. The calibration process takes only six points. While the calibration has been effective, it keeps no record of the adjustments made for the user or of how far from the calibration points the user was thought to be looking initially. By collecting statistical data on what calibration corrections are most often made, the system could improve itself over time. The location cursor was rather jittery. A larger position buffer and slower refresh rates could minimize this issue. Finally, the only method of verification used currently is the user reporting back if the system is working. A verification GUI could be constructed (similar to the calibration GUI) that records actual and intended position locations and reports how accurate the system is for different users for various locations on the screen.

The following GANTT chart outlines the time and work divisions necessary to complete the gaze controlled human-computer interface project by May, 2012.

| | **JAN 25th – FEB 15th** | **FEB 16th – MAR 7th** | **MAR 8th – APR 4th** | **APR 5th – APR 25th** | **APR 25th – MAY 2nd** |
|---|---|---|---|---|---|
| **Fuzzy Logic Research** | 🟨 | | | | |
| **Bright Pupil Hardware** | 🟨 | 🟨 | | | |
| **Bright Pupil Image Capture** | | 🟨 | 🟨 | | |
| **Image Processing** | | | 🟨 | | |
| **Implementation of Fuzzy Logic** | | 🟨 | 🟨 | 🟨 | |
| **System, User Interface** | | | | 🟨 | 🟨 |
| **FPGA Implementation** | | | 🟨 | 🟨 | 🟨 |

This chart was closely followed. FPGA implementation was removed. The organization and adequate pacing of the team contributed to the success of the project.

## X. APPENDIX

### WORKS CITED

[1]Zhu, Zhiwei, and Qiaang Ji. "Novel Eye Gaze Tracking Techniques Under Natural Head Movement." IEEE Transactions on Biomedical Engineering 54 (2007): 3346-2260.
[2] http://www.nlm.nih.gov/medlineplus/ency/imagepages/8867.htm
[3] http://vast.uccs.edu/~tboult/frame/Morimoto/node2.html
[4] Yoo D.H. and M. J. Chung. "A novel non-intrusive gaze estimation using cross-ratio under large head motion," Computer Vision and Image Understanding, vol. 98, no. 1, pp. 25-51, 2005.
[5] Kang, Jeffrey J., Moshe Eizenman, Elias D. Guestrin, and Erez Eizenman. "Investigation of the Cross-ratios Method for." 2-36.
[6] http://en.wikipedia.org/wiki/File:HSV_color_solid_cylinder_alpha_lowgamma.png

### WORKS CONSULTED

Nagamatsu, Kamahara, Iko, and Naoki Tanaka. "One-point Calibration Gaze Tracking Based on Eyeball Kinematics Using Stereo Cameras." Proceedings of the 2008 symposium on Eye tracking research & applications (2008): 95-98.

Yoo, Kim, Lee and Myoung Jin Chung. "Non-contact eye gaze tracking system by mapping of corneal reflections." Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference (2002) : 94-95.

### ADDITIONAL IMAGES

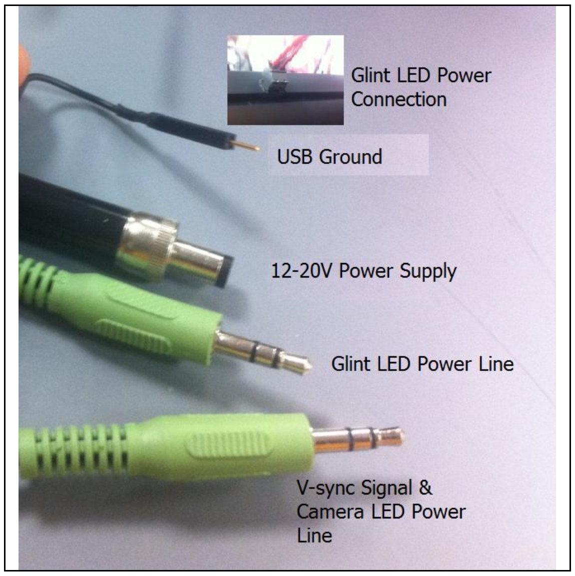

Glint LED Power Connection
USB Ground
12-20V Power Supply
Glint LED Power Line
V-sync Signal & Camera LED Power Line

Figure A1: Connectors used in the system

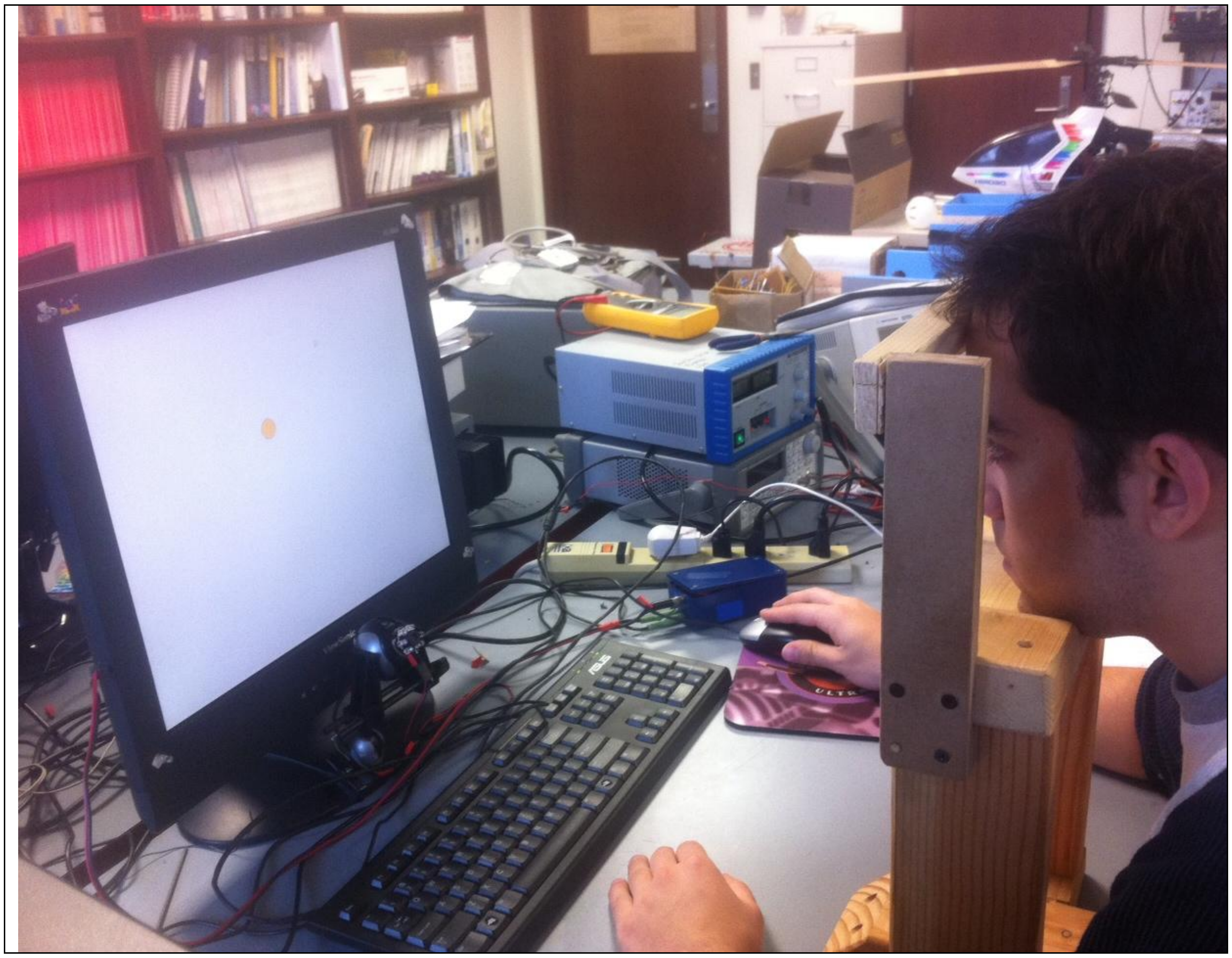


Figure A2: User interface

## SOFTWARE CODE PART I: PYTHON

```
import sys
sys.path.append("C:\OpenCV2.2\Python2.7\Lib\site-packages")
import cv
import numpy as np
import time, string
import copy
import math
from Tkinter import *
from math import*
import time
from VideoCapture import Device
class GUI:
    def __init__(self, master):
        root.done =0;

        #Tk Interface
        self.tk = Tk()
        #Program Variables


        self.state=1
        self.k=1
        self.recab = 1
        self.done = 0

        #Creating Window and Packing
        self.canvas = Canvas(master, width = w, height = h)
        self.text = self.canvas.create_text(w/2,h/2, text="Look at calibration
point and press return 3x.", fill="black", font=("Helvectica", str(20)))
        self.r=self.canvas.create_oval(0, 0, 1, 1, fill="cyan")
        self.circ = self.canvas.create_oval((4*w/5)-(rad), (h/5)-(rad),
(4*w/5)+(rad),(h/5)+(rad),fill="red")
        self.canvas.pack()

        #Bindings
        self.tk.bind("<Escape>", self.end)
        self.tk.bind("<Key-Return>", self.increase)
        self.tk.bind("<BackSpace>", self.kill)
    def increase(self,event):
        #print(avgbufx, avgbufy)
        if self.recab ==1:
            if not(self.state >=7):
                if (self.k ==1):
                    self.canvas.itemconfig(self.text, text = "Look at
calibration point and press return 3x.", fill = "black")
                    self.x1 = self.Xr*h
                    self.tx1 = self.Xr
                    self.y1 = self.Yr*w
                    self.ty1 = self.Yr
                if (self.k ==2):
                    self.x2 = self.Xr*h
                    self.tx2 = self.Xr
                    self.y2 = self.Yr*w
                    self.ty2 = self.Yr
                if (self.k ==3):
                    self.x3 = self.Xr*h
                    self.tx3 = self.Xr
                    self.y3 = self.Yr*w
                    self.ty3 = self.Yr
                self.k +=1
                if (self.k==4):
                    #print(self.x1, self.y1, self.x2, self.y2, self.x3,
self.y3)
                    if ((abs(self.x1-self.x2)<(1.5*rangee)) and (abs(self.x1-
self.x3)<(1.5*rangee)) and (abs(self.x2-self.x3)<(1.5*rangee)) and
(abs(self.y1-self.y3)<(1.5*rangee)) and(abs(self.y3-self.y2)<(1.5*rangee))
and(abs(self.y1-self.y2)<(1.5*rangee))):
                        if ((abs(((self.x1+self.x2+self.x3)/3)-
self.limx)>(5*rangee)) or (abs(((self.y1+self.y2+self.y3)/3)-
self.limy)>(5*rangee))):
```

```
                            self.canvas.itemconfig(self.text, text = "You are
not looking at circle. Recalibrate.", fill = "red")
                            self.x1 = 0.0
                            self.y1 = 0.0
                            self.x2 = 0.0
                            self.y2 = 0.0
                            self.x3 = 0.0
                            self.y3 = 0.0
                            self.tx1 = 0.0
                            self.ty1 = 0.0
                            self.tx2 = 0.0
                            self.ty2 = 0.0
                            self.tx3 = 0.0
                            self.ty3 = 0.0
                            self.recab = 0
                        else:
                            self.state+=1
                            #Rx = (self.x1+self.x2+self.x3)
                            #Ry = (self.y1+self.y2+self.y3)
                    else:
                        self.canvas.itemconfig(self.text, text = "Too much
variation. Recalibrate.", fill = "red")
                        self.x1 = 0.0
                        self.y1 = 0.0
                        self.x2 = 0.0
                        self.y2 = 0.0
                        self.x3 = 0.0
                        self.y3 = 0.0
                        self.tx1 = 0.0
                        self.ty1 = 0.0
                        self.tx2 = 0.0
                        self.ty2 = 0.0
                        self.tx3 = 0.0
                        self.ty3 = 0.0
                        self.recab = 0
                    self.k =1
                self.update()
        else:
            self.recab =1
            self.canvas.itemconfig(self.text, text = "Look at calibration
point and press return 3x.", fill ="black")

    def update(self):
        #print(xcal, ycal)
        #1
        #print(self.ty1, self.ty2, self.ty3)
        if (self.state ==1):
               self.limx = h/5
               self.limy = w/5
               self.canvas.delete(self.circ)
               self.circ = self.canvas.create_oval((4*w/5)-(rad), (h/5)-(rad),
(4*w/5)+(rad),(h/5)+(rad),fill="red")

        #2
        elif (self.state ==2):
               self.limx = h/5
```

```
                self.limy = w/2
                self.canvas.delete(self.circ)
                self.circ = self.canvas.create_oval((4*w/8)-(rad), (h/5)-(rad),
(4*w/8)+(rad),(h/5)+(rad),fill="red")
                self.calX1 = min(self.tx1,self.tx2,self.tx3)
                self.calY1 = min(self.ty1,self.ty2,self.ty3)



        #3
        elif (self.state ==3):
                self.limx = h/5
                self.limy = 4*w/5
                self.canvas.delete(self.circ)
                self.circ = self.canvas.create_oval((w/5)-(rad), (h/5)-(rad),
(w/5)+(rad),(h/5)+(rad),fill="red")
                self.calX2 = min(self.tx1,self.tx2,self.tx3)
                self.calY2 = (self.ty1+self.ty2+self.ty3)/(3.0)




        #4
        elif (self.state ==4):
                self.limx = 4*h/5
                self.limy = w/5
                self.canvas.delete(self.circ)
                self.circ = self.canvas.create_oval((4*w/5)-(rad), (4*h/5)-
(rad), (4*w/5)+(rad),(4*h/5)+(rad),fill="red")
                self.calX3 = min(self.tx1,self.tx2,self.tx3)
                self.calY3 = max(self.ty1,self.ty2,self.ty3)
        #5
        elif (self.state ==5):
                self.limx = 4*h/5
                self.limy = w/2
                self.canvas.delete(self.circ)
                self.circ = self.canvas.create_oval((4*w/8)-(rad), (4*h/5)-
(rad), (4*w/8)+(rad),(4*h/5)+(rad),fill="red")
                self.calX4 = max(self.tx1,self.tx2,self.tx3)
                self.calY4 = min(self.ty1,self.ty2,self.ty3)

        #6
        elif (self.state ==6):
                self.limx = 4*h/5
                self.limy = 4*w/5
                self.canvas.delete(self.circ)
                self.circ = self.canvas.create_oval((w/5)-(rad), (4*h/5)-(rad),
(w/5)+(rad),(4*h/5)+(rad),fill="red")
                self.calX5 = max(self.tx1,self.tx2,self.tx3)
                self.calY5 = (self.ty1+self.ty2+self.ty3)/(3.0)
        #7
        elif (self.state ==7):
                self.canvas.delete(self.circ)
                self.canvas.delete(self.text)
                self.calX6 = max(self.tx1,self.tx2,self.tx3)
                self.calY6 = max(self.ty1,self.ty2,self.ty3)
                self.done = 1
                self.XX1 = ((self.calX1+self.calX2+self.calX3)/3.0);
                self.XX2 = ((self.calX4+self.calX5+self.calX6)/3.0);
                self.YY1 = ((self.calY1+self.calY4)/2.0);
                self.YY2 = ((self.calY3+self.calY6)/2.0);




    #Functions
    def end(self, event):
        root.destroy()
        quit()

    def kill(self):
        self.canvas.delete(self.r)

    def draw(self):
        topx=ratio[1]*w+size
        botx=ratio[1]*w-size
        topy=ratio[0]*h+size
        boty=ratio[0]*h-size
##        if (self.GUIstart ==0):
        self.r=self.canvas.create_oval(topx, topy, botx, boty, fill="orange")
##        else:
##            self.r=self.canvas.create_oval(topx, topy, botx, boty,
fill="blue")

root = Tk()
w, h = root.winfo_screenwidth(), root.winfo_screenheight()
rangee = h/15
rad = h/30
##limx = h/5
##limy = w/5
root.overrideredirect(1)
root.geometry("%dx%d+0+0" % (w, h))


size=10
a=GUI(root)

try:
    while 1:
        ratio=[Xr,1-Yr]
        #print(ratio)
        #print(a.calY4, a.calY1)
        a.kill()
        a.draw()
        #if (a.done):

            #print(a.calY4, a.calY1, a.calY3, a.calY6)
        root.update_idletasks() # redraw
        root.update() # process events
        #view[:,:] =0
       # img2 = cv.GetMat(img1)
```

```
        img = cv.QueryFrame(capture)
        cv.CvtColor(img ,imgGS, cv.CV_RGB2GRAY)
        #cv.ShowImage("camera", imgGS)
        img1 = cv.QueryFrame(capture)
        cv.CvtColor(img1 ,imgGS2, cv.CV_RGB2GRAY)
        cv.Sub(imgGS, imgGS2, imgGSS)
        cv.Sub(imgGS2, imgGS, imgSGS)
        sub1 = cv.GetMat(imgGS)
        sub2 = cv.GetMat(imgGS2)

        if (np.mean(dummy) > 6):
            cv.Sub(imgGS2, imgGS, imgGSS)
            cv.Sub(imgGS, imgGS2, imgSGS)
            #sub11 = cv.CloneMat(sub1)
            k = abs(k-1)
        if (k ==0):
            sub = np.subtract(sub1, sub2)
            cv.Copy(imgGS, T)
            k = 1

        else:
            sub = np.subtract(sub2, sub1)
            cv.Copy(imgGS2, T)

        sub = cv.fromarray(sub)

        #threshold image
        im_bw = cv.CreateImage(cv.GetSize(imgGSS),cv.IPL_DEPTH_8U,1)
        cv.Threshold(imgGSS, im_bw, 50, 255, cv.CV_THRESH_BINARY) #|
cv.CV_THRESH_OTSU);
        cv.Smooth(im_bw, im_bw, cv.CV_MEDIAN, 19, 19)


        imbw_2 = (np.asarray(cv.GetMat(im_bw)))



        #find pupil center
        white_array = np.where(imbw_2 > 0)
        X_C = np.average(white_array[0])
        Y_C = np.average(white_array[1])
        X_C =np.rint(X_C)
        Y_C =np.rint(Y_C)
        if X_C>440:
            X_C=440
        if X_C<40:
            X_C=40
        if Y_C>600:
            Y_C=600
        if Y_C<40:
            Y_C=40

        if not (math.isnan(Y_C)):
            YP = int(Y_C)
        if not (math.isnan(X_C)):
```

```
    XP = int(X_C)
YP = int(YP)
XP = int(XP)

#print "(" + str(q) + ", " + str(p) + ")"

#display black dot at pupil cetroid
if (X_C > 0 and Y_C > 0):
    imbw_2[X_C][Y_C] = 0


#print (X_C, Y_C)

U = copy.deepcopy(np.asarray(cv.GetMat(T)))

U = cv.fromarray(U)

XPmin=XP-30
YPmin=YP-30
XPmax=XP+30
YPmax=YP+30

#print(XPmin,  XPmax, YPmin,YPmax)

#unsliced_arr = (np.asarray(cv.GetMat(U)))
slice = U[ XPmin:XPmax,YPmin:YPmax ]

P = copy.deepcopy(np.asarray(cv.GetMat(slice)))
P = cv.fromarray(P)

slice_bw = cv.CreateImage(cv.GetSize(P),cv.IPL_DEPTH_8U,1)
cv.Threshold(P, slice_bw, 150, 255, cv.CV_THRESH_BINARY)
cv.Smooth(slice_bw, slice_bw, cv.CV_MEDIAN, 3, 21)

slice = slice_bw[6:36,5:35]
P1 = copy.deepcopy(np.asarray(cv.GetMat(slice)))
P11 = copy.deepcopy(P1)
P1 = cv.fromarray(P1)
P11 = cv.fromarray(P11)

slice = slice_bw[26:56,5:35]
P2 = copy.deepcopy(np.asarray(cv.GetMat(slice)))
P22 = copy.deepcopy(P2)
P2 = cv.fromarray(P2)
P22 = cv.fromarray(P22)

slice = slice_bw[6:36,26:56]
P3 = copy.deepcopy(np.asarray(cv.GetMat(slice)))
P33 = copy.deepcopy(P3)
P3 = cv.fromarray(P3)
P33 = cv.fromarray(P33)

slice = slice_bw[24:54,25:55]
```

```
P4 = copy.deepcopy(np.asarray(cv.GetMat(slice)))
P44 = copy.deepcopy(P4)
P4 = cv.fromarray(P4)
P44 = cv.fromarray(P44)

p1 = (np.asarray(cv.GetMat(P11)))
p2 = (np.asarray(cv.GetMat(P22)))
p3 = (np.asarray(cv.GetMat(P33)))
p4 = (np.asarray(cv.GetMat(P44)))

k1 = np.average(p1)
k2 = np.average(p2)
k3 = np.average(p3)
k4 = np.average(p4)

if k1>0:
    q1 = np.where(p1 > 0)
    x1 = (q1[0].max())
    y1 = (q1[1].max())
    if x1 >= 22.0:
        x1 = (q1[0].min())
    x1 = 31.0-x1
    if y1 >= 23.0:
        y1 = (q1[1].min())
    y1 = 31.0-y1
if k2>0:
    q2 = np.where(p2 > 0)
    x2 = (q2[0].min())
    y2 = (q2[1].max())
    if x2 <= 8.0:
        x2 = (q2[0].max())
    x2 = x2+1.0
    if y2 >= 23.0:
        y2 = (q2[1].min())

    y2 = 31.0-y2

if k3>0:
    q3 = np.where(p3 > 0)
    x3 = (q3[0].max())
    y3 = (q3[1].min())

    x3 = 31.0 - x3
    y3 = y3+1.0

if k4>0:
    q4 = np.where(p4 > 0)
    x4 = (q4[0].min())

    y4 = (q4[1].min())

    if x4 <= 8.0:
        x4 = (q4[0].max())
    x4 = 1.0+x4
```

```
        if y4 <= 7.0:
            y4 = (q4[1].max())
        y4 = 1.0+y4




    XH = (x1)
    XL = (x2+x4)/2.0
    YH = (y1+y2)/2.0
    YL = (y4)




    Xr = (XH/(XL+XH))
    Yr = (YH/(YL+YH))
    #print(Xr, Yr)

    Yr = ((Yr-0.2)*(1/(.8-.2)))
    Xr = ((Xr-0.23)*(1/(.7-.23)))

    #print(Yr)

    if (a.done==1):
        Yr = ((0.6/(a.YY2-a.YY1))*(Yr-a.YY1)+0.2)
        Xr = ((0.6/(a.XX2-a.XX1))*(Xr-a.XX1)+0.2)








    if Xr>=1.0:
        Xr = 1.0

    if Yr>=1.0:
        Yr = 1.0

    if Xr<=0.0:
        Xr = 0.0

    if Yr<=0.0:
        Yr = 0.0




    if (a.done==1):
        bufx1 = Xr
        bufy1 = Yr
```

```
            Xr =
(bufx1+bufx2+bufx3+bufx4+bufx5+bufx6+bufx7+bufx8+bufx9+bufx10)/10.0
            Yr =
(bufy1+bufy2+bufy3+bufy4+bufy5+bufy6+bufy7+bufy8+bufy9+bufy10)/10.0
            bufy2 = bufy1
            bufy3 = bufy2
            bufy4 = bufy3
            bufy5 = bufy4
            bufy6 = bufy5
            bufy7 = bufy6
            bufy8 = bufy7
            bufy9 = bufy8
            bufy10 = bufy9

            bufx2 = bufx1
            bufx3 = bufx2
            bufx4 = bufx3
            bufx5 = bufx4
            bufx6 = bufx5
            bufx7 = bufx6
            bufx8 = bufx7
            bufx9 = bufx8
            bufx10 = bufx9




        #print(ratio)

        xcor = Xr*959.0
        ycor = 1279.0-Yr*1279.0






        #images to display
        #cv.ShowImage("camera1", imgGS)
        #cv.ShowImage("camera2", imgGS2)
        #cv.ShowImage("camera3", im_bw)
        #cv.ShowImage("camera4", slice_bw)
    #UPDATING THE VARIBALE FOR fifo
        #if (count<=15):
         #   count +=1
        #else:





        diffsqx1 = abs(Xr-bufx1)
        diffsqx2 = abs(Xr-bufx2)
        diffsqx3 = abs(Xr-bufx3)
        diffsqx4 = abs(Xr-bufx4)
        diffsqx5 = abs(Xr-bufx5)
        diffsqx6 = abs(Xr-bufx6)
        diffsqx7 = abs(Xr-bufx7)
        diffsqx8 = abs(Xr-bufx8)
        diffsqx9 = abs(Xr-bufx9)
        diffsqx10 = abs(Xr-bufx10)

        diffsqy1 = abs(Yr-bufy1)
        diffsqy2 = abs(Yr-bufy2)
        diffsqy3 = abs(Yr-bufy3)
        diffsqy4 = abs(Yr-bufy4)
        diffsqy5 = abs(Yr-bufy5)
        diffsqy6 = abs(Yr-bufy6)
        diffsqy7 = abs(Yr-bufy7)
        diffsqy8 = abs(Yr-bufy8)
        diffsqy9 = abs(Yr-bufy9)
        diffsqy10 = abs(Yr-bufy10)



        variancex =
max(diffsqx1,diffsqx2,diffsqx3,diffsqx4,diffsqx5,diffsqx6,diffsqx7,diffsqx8,d
iffsqx9,diffsqx10)
        variancey =
max(diffsqy1,diffsqy2,diffsqy3,diffsqy4,diffsqy5,diffsqy6,diffsqy7,diffsqy8,d
iffsqy9,diffsqy10)
        variancex = (variancex)*15 +0.75
        variancey = (variancey)*15 +0.75

        if (a.done==1):
            certainty = max(variancex, variancey)
        else:
            certainty = 1
        #if count >=16:
        size = 10*certainty
        print (certainty, bufy10)



        #count = 1

        a.Yr = Yr
        a.Xr = Xr

        #print(Xr, Yr)

        if cv.WaitKey(10) == 27:
            break
except TclError:
    pass # to avoid errors when the window is closed
```

## SOFTWARE CODE PART II: ARDUINO

## DATASHEETS OF COMPONENTS USED