Lassonde School of Engineering, York University

ENG 4000

Final Design Report

T B D

Eye-Tracking Computer Human Interface Device for Disabled Individuals

Project Members

Josh Sideris

Prasanthan Urithirakodeeswaran

Ragavan Thurairatnam

Shailja Sahani

Faculty Advisor: John Tsotsos

Course Director: Michael Jenkin

Date: April 29 2013

Abstract

This final design review report provides a complete analysis of Technology Benefiting the Disabled's Eye-Tracking Computer Human Interface Device for Disabled Individuals. Individuals with physical disabilities will be able to use their personal computers through their vision using our eye-tracker. Our group, Technology Benefiting Disabled (TBD), is focused on the disabilities that limit upper body function, such as severe motor neuron disease (or otherwise some debilitating injury causing long-term confinement to a wheelchair) and implemented a solution that allows eye–tracking to assist in human–to–computer interaction. To track the user's eye movements resulting in cursor control on the screen by modifying the Starburst algorithm that is a traditional eye tracking method. This report showcases how we are able to develop an eye tracking solution for a few hundred dollars whereas the commercial solutions cost approximately $5000 to $30000 without compromising on the performance of the system.

The report begins with a description of its purpose, problem, and scope. Next it provides theoretical background on the device's operation. The main discussion deals with the design decisions made to maximize performance. The work done was on budget and on time. Initial prototypes proved eye tracking is a feasible option for on screen cursor control. Tests results have indicated this product is a viable option for commercial use.

Acknowledgments

We would like to acknowledge contributions of Professor John Tsotsos, who generously offered us the use of his laboratory facilities. Also, we would like to thank Stnaley Lio for his excellent photography skills, Ann Bublitz and Christian Haas for giving us a space to work in, Patrick Irvin for letting us use his tools, Shah Sotodeh for testing our solution, Guy Benari for demonstrating the solution, and the developers of the Starburst algorithm. Moreover, we would like to acknowledge the Physical, Sensory & Medical Disability Services for partnering with us to test our technology and providing us with a wheelchair for demonstration. Finally, we are most thankful to Michael Jenkin for his continued insight and patience.

Notation List

AR: Augmented Reality

ArUco: AR Library based on OpenCV

CDR: Critical Design Report

CU: Cursor Updater

dpi: Dots Per Inch

FDR: Final Design Report

HID: Human Interface Device

IR: Infrared

OpenCV: Open Source Computer Vision Library

PDR: Preliminary Design Report

RANSAC: Random sample consensus

TBD: Technology Benefiting the Disabled

USB: Universal Serial Bus

Table of Contents

1 INTRODUCTION **1**

1.1 PURPOSE **1**
1.2 BACKGROUND **1**
1.3 SCOPE **1**

2 TECHNICAL DESCRIPTION **2**

2.1 USER ROLES **2**
2.2 HARDWARE **2**
2.2.1 ARCHITECTURE 2
2.2.2 HEAD MOUNT 3
2.2.3 CAMERAS 4
2.3 SOFTWARE **4**
2.3.1 ARCHITECTURE 4
2.3.2 LIMBUS DETECTION ALGORITHM 5
2.3.3 CLICK INPUT 7
2.3.4 SCENE TO SCREEN MAPPING 8
2.4 INTEGRATION **11**
2.4.1 COMPUTER REQUIREMENTS 11
2.4.2 USE 11

3 DESIGN DECISIONS, OBSTACLES & RISKS **12**

3.1 IMPLEMENTING/OPTIMIZING THE LIMBUS DETECTION ALGORITHM **12**
3.1.1 OBSTACLES AND OPTIMIZATION EFFORTS 13
3.2 IMPLEMENTING SCENE TO SCREEN MAPPING **14**
3.2.1 IMPLEMENTING AUTO-CURSOR POSITIONING IN WINDOWS 14
3.3 BUILDING A PROTOTYPE TO TEST THE SOFTWARE **14**

4 PERFORMANCE, OPTIMIZATION & TESTING **15**

4.1 PERFORMANCE **15**
4.2 ACCURACY TESTING **18**

5 DEMONSTRATION & APPLICATION **20**

6 FUTURE WORK **20**

6.1 GUIDED ADJUSTMENT **20**
6.2 HEAD MOUNT **20**

7 BUDGET **21**

8 IMPACT **22**

9 CONCLUSION **22**

APPENDIX A: HARDWARE SPECIFICATION **I**

APPENDIX B: CAMERA SPECIFICATION **II**

REFERENCES **III**

CONTRIBUTION SHEET **IV**

Table of Figures

FIG I: Eye Tracker Set-up 3
FIG II: Camera Set-up 4
FIG III: Software Architecture 5
FIG IV: Eye Features Recorded 5
FIG V: Ellipse Fitting on the Eye 6
FIG VI: Glyphs Used 8
FIG VII: Scene Cartesian Coordinates 9
FIG VIII: Scene Coordinate Intersection 9
FIG IX: Scene Trapezoid Coordinates 10
FIG X: Scene Calculation 10
FIG XI: Eye tracker in use 12
FIG XII: Time to execute vaious RANSAC iterations 16
FIG XIII: Execution Time for RANSAC vs. Threads per Block (per 1 Block) 17
FIG XIV: Execution Time for RANSAC vs Blocks Issued 17
FIG XV: Execution Time per 7 Blocks 18
FIG XVI: Directional Accuracies Sampled 18
FIG XVII: Future Hardware Prototype 20

Table of Tables

Table I: Performance Requirements 7
Table I: Budget 21
Table III: Contributions IV

1 Introduction

1.1 Purpose

This report documents the final design by Technology Benefiting the Disabled (TBD) on the Eye-Tracking Computer Human Interface Device for Disabled Individuals to its graded Critical Design Review (CDR) report of April 24th 2013. A list of tasks completed, for the final demonstration on May 1st 2013, is included in this report.

1.2 Background

Traditional ways of using a computer cannot be used by everyone. In fact, one in nine disabled Canadians is affected by mobility[1]. This provided an opportunity to assist a large portion of the population by leveraging technology. At its current state, computers require fine motor skills whether it is for touch, type or click. Instead, our computing interface tracks the user's eyes allowing hands-free control of the cursor.

Using this solution allows persons with physical disabilities to use a computer with similar level of control as a user without said disabilities. Ideally, this solution is usable by anyone to interface with a computer.

1.3 Scope

This report provides the status of all tasks described in the graded CDR, which include the following:

1. Implementing the limbus detection algorithm
2. Implementing Scene to Screen Mapping
3. Implementing auto-cursor positioning in Windows
4. Improving the prototype to test the software
5. Performing testing and optimizations
6. Building an application to test the solution

2 Technical Description

The eye tracker is a device that allows for cursor control using eye movements. This is done by detecting the user's eye shape and converting this to an estimate of cursor position using existing software algorithms modified by TBD. First, the solution uses the Starburst [2] algorithm to fit an ellipse to the eye camera image, allowing for an estimation of limbus center where the corneal limbus, or limbus, is the border of the cornea. The ellipse fitting is done through RANSAC [3] iterations. Once this is done, a secondary camera, pointed towards the screen, is used to map the estimated gaze point onto the user's view while using the user's position relative to the display.

The product will require the outside intervention of a caregiver to place the device on the user and calibrate the user's eye. After the setup is completed, the user is able to use the computer as desired. TBD is using a common Windows application to demonstrate the use of the eye-tracker, as they are readily available on most personal computers. The major components needed to operate the solution include user roles, hardware, and software.

2.1 User Roles

The main user roles needed to operate our eye-tracker are a *caregiver* and a *user*. A caregiver needs to be present to assist the user in setting up and calibrating the device. The caregiver must be present in order to fit the unique shape of the user's limbus using our eye-tracker. While using the initial prototypes, we found that adding these user roles were necessary in order to use this device properly.

Initially, the caregiver positions the solution's hardware and runs the calibration software for the user. Once this step is complete, the use is free to navigate on their personal computer using their vision instead of a mouse.

2.2 Hardware

2.2.1 Architecture

In this eye-tracker, hardware architecture refers to the carefully designed structure of the device using a head mount and cameras' placements to be used by the user. The purpose

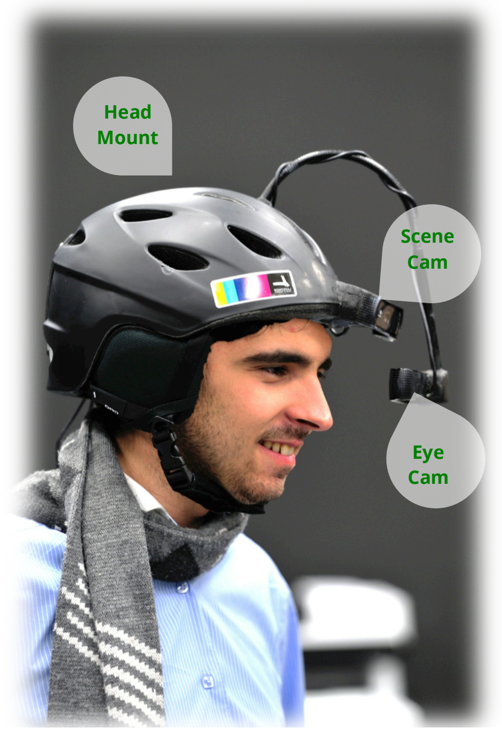


FIG I: Eye Tracker set-up

of designing the hardware is to produce a prototype that can be used easily by any user. This design includes two cameras secured to the head mount. The camera directions vary as the eye-cam points towards the user's eye and the scene-cam points towards the user's view. The current head mount is shaped like a helmet to be worn on the user's head, as seen in FIG I.

The hardware architecture has evolved from the first prototype to reduce errors and increase the user experience. The first design was inaccurate and unstable and only used one camera. The second design is much more stable, visible in FIG I, making use of a secure and stable platform. The third iteration of this design shall be refined to reduce weight and increase comfort.

The major components visible to the user and the caregiver are the head mount and cameras.

2.2.2 Head Mount

The *head mount* refers to the platform placed in its operating position - to secure the cameras on the user's head. The benefits of having a head mount include synchronization of the head movements with the eye-tracker system. A head mount's purpose is simply to hold the camera positions in place while the solution is in use.

TBD's first head mount was made using a baseball hat, and we quickly realized this solution doesn't provide the desired stability. The current head mount is an off-the-shelf generic snowboarding helmet and a complete profile of this helmet is available in APPENDIX A. This helmet is black, medium in size, with plastic casing on the outside and foam padding on the inside. It also has ventilation holes that we use to secure the cameras' positions in place and these holes also regulate the temperature of our user's head.

2.2.3 Cameras

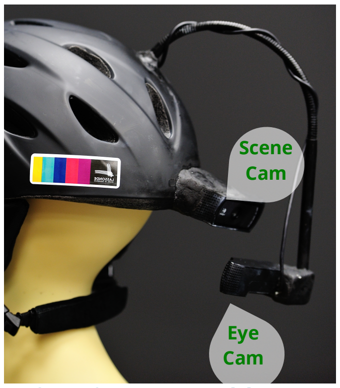


FIG II: Camera Positions

The *cameras* are used to record images of interest to the eye tracker in real time. These cameras record the user's eye using the *eye cam* and the view using the *scene-cam*. The cameras haven't changed because they are lightweight, small and can be modified to fit the head mount. The model number is Logitech HD Pro Webcam c920 and these cameras record in High Definition (HD) in the visible light spectrum. A complete specification is available in APPENDIX B.

The camera's external casing is a black plastic, with a glass front and it has a USB wire extending from the rear. Originally TBD intended to modify them by adding an IR pass filters inside but we are now working with them without any internal modifications.

The reasons for changing the design were mostly due to construction, maintainability and safety. The infrared system requires circuitry to be soldered, which creates more points for the system to break. It also requires a separate power source: if a battery were used, this would have to be replaced periodically, otherwise modifications to the off shelf camera would have to be made to use power from it. By using a visible light solution, we can avoid having to open up the cameras to change the filters and avoid potential damage to the camera. This also means the hardware is more reliable, since there are fewer components that could fail (LED, circuitry, etc.). This also removes the possibility of the Infrared light potentially damaging someone's eye if it were to get too close. We also don't have to mount the LED to the camera, which can be bent causing issues. Another advantage to the visible light method is that we won't get interference from other infrared sources.

2.3 SOFTWARE

2.3.1 Architecture

The software architecture is a modular design connecting the software components required to map eye movements on screen. The modular nature of this design allowed parallel development and modifications as necessary. The core structure of the software is depicted in the FIG II and has changed to include the scene camera's transformation from scene to screen coordinates.

A simple protocol is used between all components, visible by the flow diagram in FIG III. The primary service, the eye-tracking algorithm, is responsible for measuring the users gaze and projecting it onto the scene image in terms of x and y coordinates relative to the captured image. Next, the scene to screen transform (mapping) component takes these continuous x and y coordinates and calculates the gaze of the user relative to the screen he or she is looking at. This is fed into API calls from the Win32 library to set the cursor location, which completes the function of moving the cursor to where the user is looking.

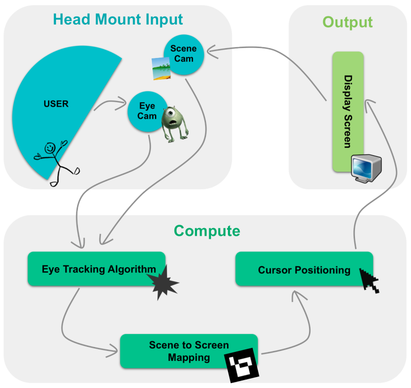


FIG III: Software Architecture

2.3.2 Limbus Detection Algorithm

There are a large variety of methods that can be used to detect the gaze of an eye. Among these is the Starburst algorithm[2], which we have chosen for its simplicity, accuracy, and speed. Starburst was originally designed to detect the pupil center of an eye using infrared lighting to illuminate a reflection in the cornea. However, we are using a modified version, which detects the limbus under direct, visible light.

First, the current limbus center is estimated, either by selecting the center pixel, or by using the previous limbus center if available. Rays are sent out radially from this point in order to find a threshold - that is, a point where the intensity varies sharply. When an edge is found it is added as a feature point. If the candidates are not valid, they will not match an ellipse. The edge detection process is then repeated starting from the average of the candidate points. This successively gets closer to the center of the limbus, and once the change in pupil center is less than 10 pixels. If no center is found within ten iterations then we assume this is a blink/bad frame.

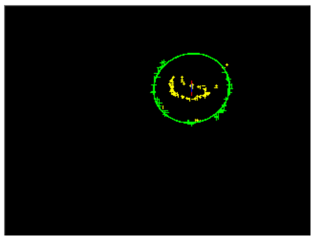


FIG IV: Eye features recorded

Random sample consensus or RANSAC[3], an algorithm to perform fitting of models when data contains outliers, is then used to remove outliers from the candidates of the limbus contour. Inliers are chosen based on the distance to the ellipse compared to a threshold. The threshold is determined from a probabilistic model. The original Starburst algorithm uses a threshold of 1.98 pixels.

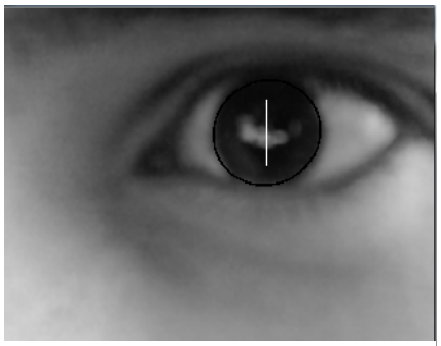


FIG V: Ellipse fitting on the eye

The mapping of the eye position to the scene is determined by the homography between the positions of the limbus center and on the screen. The calibration data is obtained by the having the user looking nine known points on the screen and recording the eye positions. The calibration is used to create the mapping matrix, which is applied to the eye coordinates, which results in screen coordinates that the user is looking at.

2.3.2.1 Performance Requirements

The minimum requirement for use is to allow the user to press large on screen buttons with their eyes. This would allow them to use basic accessible programs to do a simple computer task, such as reading an electronic document. For example, Adobe Acrobat's next page button is approximately 40 x 40 pixels. Thus, we would want our accuracy to be within 40 pixels to allow the user to select this button.

We also want the user to sit comfortably away from the screen. During testing we found 20 inches to be a comfortable distance from a 72 dpi screen. From this, we can calculate the minimum error in degrees needed to execute this task.

$$Arctan\left(\frac{\frac{40\,pixels}{72\,dpi}}{20\,inches}\right) = 1.59\,degrees$$

The Starburst algorithm ideally results in 1-degree error, so it will allow us to meet our requirement. In testing, we have achieved lower errors (this is described in the testing section).

The update rate requirement is based on the restrictions of the camera. The camera's max frame rate is 30 frames per second. Thus we want to be able to update the screen at approximately 30 updates a second. This is affected by the rate at which we can process images, which is calculated in section 4 of the report. The delay threshold requirement is calculated in section 4.2. An overview of these requirements, shown in Table I, drives these performance improvements of our system.

Table I: Performance Requirements

Variable	Required
Directional Accuracy	1.59 degrees
Update Rate	~30 iterations/second
Eye Cam Update	33 milliseconds
Delay Threshold	133 milliseconds

2.3.3 Click Input

Clicking allows access to many actions on a computer, however since the mouse has been replaced with the eye tracker, there is no mouse button to press. The early iterations of the eye tracker used a large, easily accessible button, designed for users with at least some motor control in their hands, if not fine motor control.

This was then replaced by the use of an extended blink from the user to input a mouse click. This means when the user closes their eye for a long period of time, beyond that of the average eye blink (0.1-0.4 seconds [3] this will register as a click event. To prevent accidental clicks from regular blinking, we choose the threshold to register a click as twice the average (0.8 seconds). However, this method was deemed unsuitable as it did not feel natural and strained the eyes.

We replaced the mechanism for clicking with a voice recognition application that is built into Windows. By using a microphone the user can initiate various mouse actions with words. For example, the user can say the words "click" or "right click" to replace the physical clicking mechanisms of the mouse. This method provides a more natural method of clicking compared to the extended blink previously considered. Although not required, training the voice recognition system with test phrases allows for increased accuracy.

2.3.4 Scene to Screen Mapping

Using the modified Starburst algorithm explained above, we use the center of the user's eye to determine where they are looking in the world (also known as the image acquired from the scene camera). The next stage in the pipe is to determine whether the user's gaze point falls onto the connected computer monitor, and if so, move the cursor to the position where the user is looking.

Scene to screen mapping refers to the process by which our software will determine the coordinate system of the screen with respect to the user. The problem being addressed is that of the initial calibration becoming invalid if the user moves because different viewpoints have different perspectives of the scene. Reconstruction of the screen coordinate system is required in order to ensure that the user's gaze vector will continue to project onto the correct location on the screen even when the user is moving around.

In earlier versions of this project, we drew colored boxes in the corners of the screen in order to identify the location of the monitor. However, this method is very distracting to an individual trying to use the PC. A better, less distracting solution was needed.

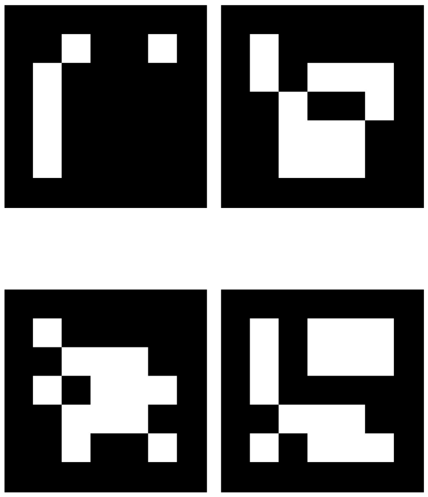


FIG VI: Glyphs used

Our goal was to use computer vision on the image acquired by the scene camera to construct a coordinate system for the monitor as computationally fast, efficient, and nonintrusive as possible. The method we decided on was to use computer vision glyphs (defined as a small modular two-dimensional pattern that is easily recognizable using computer vision techniques, shown right).

Glyph[4] detection is a relatively trendy problem in computer science, and there are some open source libraries based on OpenCV[5] provide a framework for glyph detection. For our prototype we used the used the ArUco[6] library for glyph detection.

Behind the scenes, ArUco uses OpenCV's rectangle detection to find possible candidates for glyphs (as a side note, a 3D rectangle may me transformed in any number of affine ways – OpenCV's rectangle detection accounts for this). Second, a fine-grain analysis is used to identify an integer code each glyph represents (there are around 2000 combinations of glyphs possible with this library).

We put glyphs near the four corners of the monitor, and used our ArUco glyph detection do get the coordinates of each corner of each glyph. This gives us a trapezoidal shape on the scene image representing the rectangular surface of the monitor. The trapezoid in combination with the coordinates of the user's gaze can be used to determine where to position the cursor.

Given a trapezoid representing the monitor defined by four points on a Cartesian plane, and a point representing user's gaze on the same Cartesian plane, we can construct a Cartesian plane on the trapezoid, and map the gaze point onto that plane.

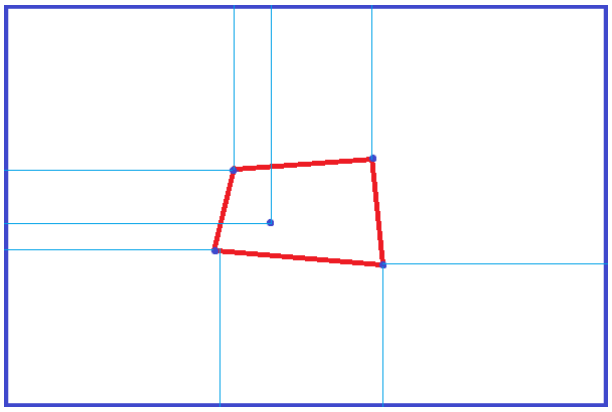


FIG VII: Scene Cartesian Coordinates

First, the intersection of two opposite sides of the trapezoid is found and a line is drawn between the intersection and gaze point. The intersection can be found using the line intersect equation. If the opposite sides are parallel, the line chosen will also be parallel and intersect with the gaze point.

$$(P_x, P_y) = \left(\frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \right)$$

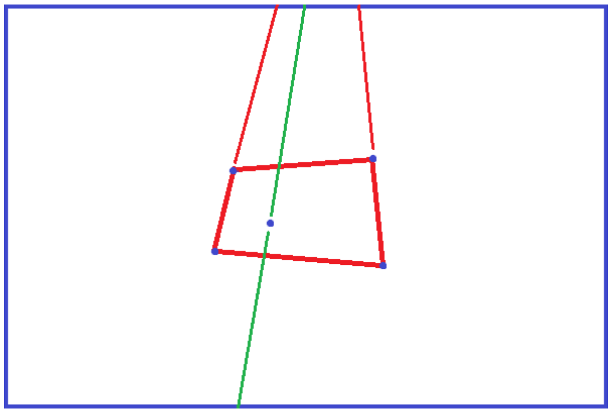


FIG VIII: Scene Coordinate Intersection

The intersection between the newly calculated line and the other two sides of the trapezoid (adjacent to the sides previously intersected) must be found.

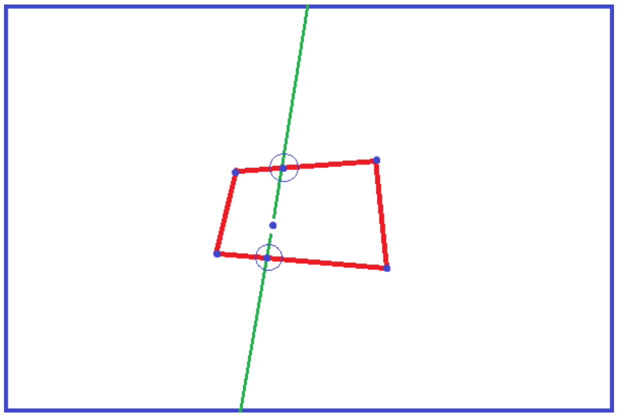


FIG IX: Scene Trapezoid Coordinates

The distance between the gaze point and either the top or bottom intersect points divided by the distance between the two intersection points can be multiplied by the screen height to get the distance from the top of the screen or the bottom of the screen, respectively.

The same calculation can be done horizontally to get the distance from the left or right side of the screen. This can be modeled by the following process.

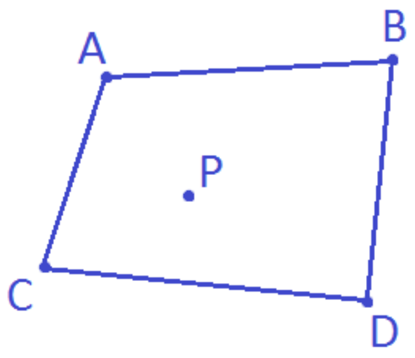


FIG X: Scene calculation

$$\begin{aligned}
E &= Intersect(A-C, B-D)\\
TopI &= Intersect(A-B, E-P)\\
BottomI &= Intersect(C-D, E-P)\\
\\
F &= Intersect(A-B, C-D)\\
LeftI &= Intersect(A-C, F-P)\\
RightI &= Intersect(B-D, F-P)\\
\\
ScreenHeight &= BottomI_y - TopI_y\\
ScreenWidth &= RightI_x - LeftI_x\\
\\
MappedP_y &= (P_y - TopI_y)\,/\,ScreenHeight\\
MappedP_x &= (P_x - TopI_x)\,/\,ScreenWidth
\end{aligned}$$

Where E is the intersect between the right and left edges of the monitor, and F is the intersect between the top and bottom edges of the monitor.

2.3.4.1 Enhanced glyph-based screen mapping

An obvious pitfall with the method used to map the cursor position using glyphs as described above is that it does not account for missing glyphs. The reality is that the scene camera often loses sight of glyphs. In this section we take a look at the robust, expandable algorithm used within our system to guess the position of lost glyphs.

1. Each data structure representing a glyph contains the position of each other glyph measured when both glyphs were both spotted at the same time on the scene image.
2. If a glyph is missing (we must guess it's location),
    a. Loop through each visible glyph.
    b. The data structures for the missing glyph and each non-missing glyph know the last positions of each other, when both glyphs were seen at the same time.
    c. For each non-missing glyph, the ratio between each component (x/y) of each edge (top, bottom, left, right) and their former values is taken.
    d. The ratio calculated above can be shown to be equal to the ratio between the components of the distance between the missing and current non-missing glyph along with the old components of distance.
    e. The average of each measurement is taken to estimate the new location of the missing glyphs.

2.4 INTEGRATION

2.4.1 Computer Requirements

The eye tracker requires two USB ports to be available on the computer in order for the eye and scene camera to be plugged in. A flat screen monitor is also required.
A computer with a minimum Pentium 4 processor, at least 1GB of RAM, and Windows XP or greater is heavily recommended.

2.4.2 Use

The system is designed under the assumption of a distance of roughly 50-75 cm between the user's head and the computer screen, and maintaining this distance through use is highly recommended.

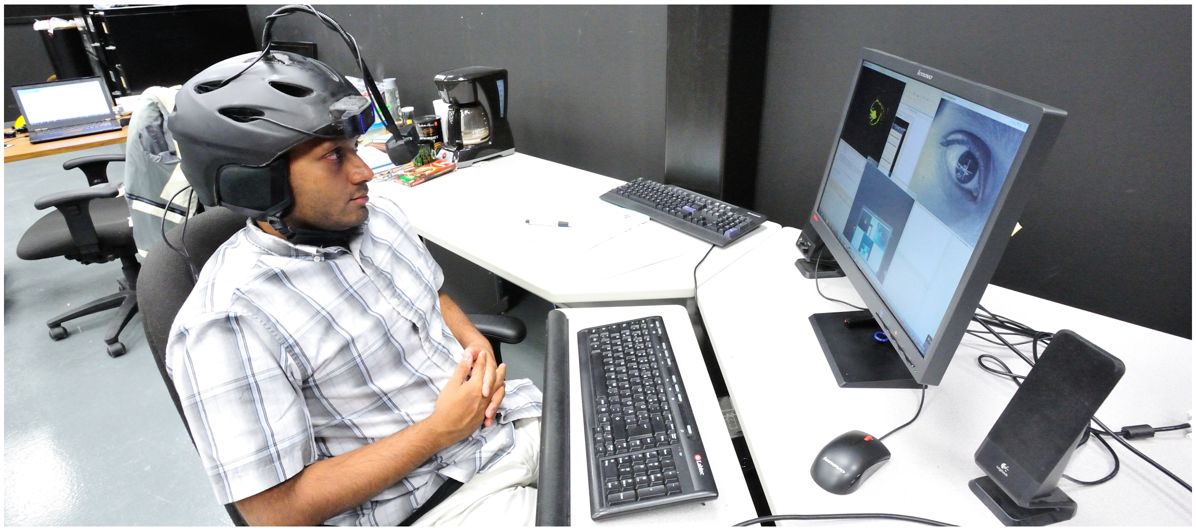


FIG XI: Eye tracker in use

3 Design Decisions, Obstacles & Risks

The design decisions with regards to the major components of our tasks mentioned in section 1.3 is as follows:

3.1 Implementing/Optimizing the Limbus Detection Algorithm

Currently the team has a working real-time implementation of the visible light tracking via modified Starburst (although the error rate has a lot of room to go down). The code dealing with infrared light has been modified to ignore corneal reflections and detect the limbus.

One disadvantage to switching is that it is harder to detect the eye without the corneal reflection. However, constraints added to RANSAC increase detection rates.

Due to the difference in position of the scene camera from the eye, they see different images from parallax. So to minimize error we need to try to keep the scene camera close to the tracked eye. Various other optimizations have been added to the eye-tracking component since the last update.

3.1.1 Obstacles and Optimization Efforts

The Starburst algorithm faces several limitations when run in real-time due to the inability to perform instantaneous fine grain filtering of the features. This results in a large source of inaccuracy due to different face shapes and different dominant features on people's faces. For example, a common source of error is the presence of dominating eyebrows causing incorrect feature detection. This inevitably leads to the ellipse being incorrectly fitted by the RANSAC algorithms, as outliers are not recognized as outliers due to their number. To prevent this behavior, various restrictions have been imposed on the RANSAC algorithm results.
These restrictions are based on working tests and have been determined through experiment.

1) ***Restrict the ranges of the limbus center***: Behind a steady mounted camera, the range of *x* and *y* coordinates the pupil center can occupy is restricted. That is, it is reasonable to assume, for example, that the limbus center will never occupy the far corners of the screen, or the far top or bottom portions of the screen. Such restrictions, when imposed on the RANSAC algorithm, prevents features such as the eyebrows, which usually occupy the extremities of the screen, from infringing too much on making the proper estimation of the ellipse.

Restrictions on 640x480 camera: $400 < x < 120$, and $380 < y < 120$

2) ***Ratios of the ellipse*****:** The angle from camera to eye is relatively low, so the primary source of distortion in the ellipse ratio (that is, the ratio of a and b parameters of the ellipse) is when the eye looks to the left and right of the screen. We can assume that certain parameters for the ellipse would never occurred in regular usage. The shape of the limbus will never be extremely wide, for example.
Furthermore, we can restrict the accepted ratios depending on the estimated position of the limbus center. As an example, if the limbus center is located close to the left portion of the screen, the ratio of the ellipse parameters cannot favor a circular ellipse.

3) ***Restrict the size of the ellipse*****:** The early calibration of the camera center ensures that the center of the eye will be in the expected spot. We also ensure that the size falls under an expected value. By formally restricting the size, we also protect against the ellipse being fitted to exterior features such as the eyebrows (which usually results in a larger ellipse being formed).

Restrictions on 640x480 camera: $92 < a < 65$ and $92 < b < 65$

3.2 Implementing Scene to Screen Mapping

The current screen mapping technique uses custom C++ code to scan the scene camera image for green or red clusters. In a separate thread, four borderless Windows form are drawn on the screen, one in each corner, and the colors are alternated between red, and green.

3.2.1 Implementing Auto-Cursor Positioning in Windows

The Win32 library provided by Microsoft makes it trivial to update cursor position via C++ code, thus this step is essentially complete and is awaiting final implementation from the eye tracker and mapping systems to be fully implemented.

3.3 Building a Prototype to Test the Software

An initial prototype was constructed using a baseball hat along with spare cameras. We soon found limitations in both the physical design as well as the cameras. The baseball hat, for example, was not nearly stable enough for the weight of the cameras, and our tests were affected by this loss of stability. To solve this we bought new Logitech C920 cameras and constructed a new head device. The new device is described at length above in the hardware section. In our construction, we have also encountered several obstacles with the current design.

a) *Weight*: The head mount, combined with two cameras, adds a great deal of weight on the head. This does introduce discomfort and, equally as important, a greater potential to cause movement and maladjustment during the use of the camera. The helmet design is not comfortable to wear for extended periods of time. The imbalance and front-heavy design is also prone to slippage, which is of course a problem for calibration mapping.

b) *Adjustment:* The current design, in order to maintain non-intrusiveness, has the camera placed farther from the eye (about ten centimeters). This introduces a difficulty when adjusting the cameras, as the camera uses digital zoom, and a small adjustment in the camera results in a major shift from the perspective of the eye camera.

c) *Visual blocking*: The camera facing the eye is mounted from an arc and comes down in front of the eye. This blocks the view of the screen partially. The ideal solution is to place the camera at an angle facing downwards, such that the entire eye can be seen without blocking the screen from view.

4 PERFORMANCE, OPTIMIZATION & TESTING

4.1 Performance

The system's accuracy and user experience is closely related to the execution speed of the system.

The performance of the system has an impact on the user experience. Since the camera's frame-rate is 30 frames per second, we're restricted by this in terms of processing the frames.

$$\frac{1\,second}{30\,frames} * \frac{1\,second}{1000\,milliseconds} = 33.3\,ms$$

Thus the total time to process the frames must be less than 33.3 milliseconds.

One of the issues with the RANSAC approach is that it varies in performance based on the iterations executed. The number of iterations varies with the number of inliers found. So when an ellipse that approximates most points of the limbus it finishes quickly, within approximately 1 millisecond. However, when the eye is not found or many features not on the eye are found, the time to execute RANSAC becomes very high.

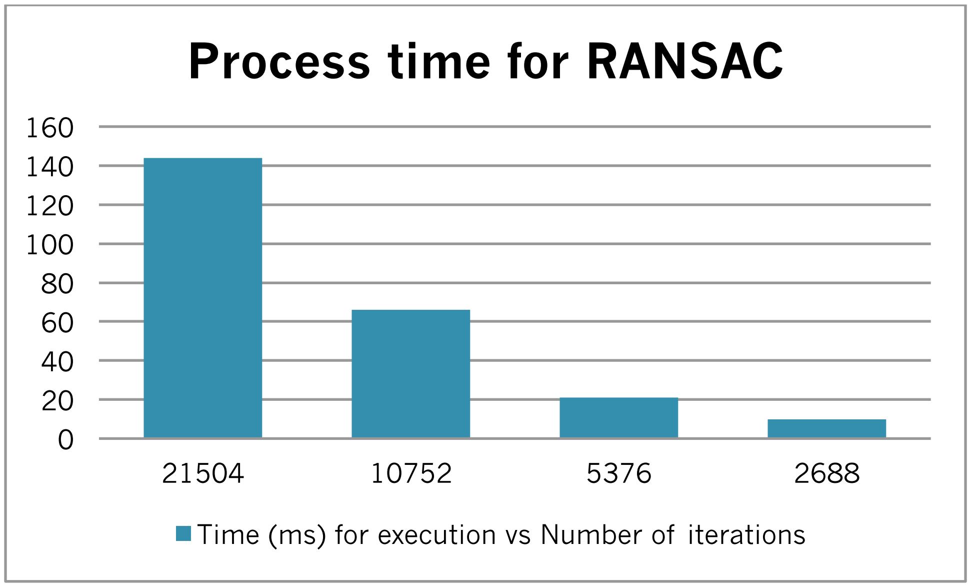


FIG XII: Time to execute vaious RANSAC iterations

If we take the worst case execution time per iteration and the required time to execute RANSAC we can find that the max acceptable RANSAC iterations.

$$Max\ Iterations\ of\ RANSAC = \frac{MaxTimeToProcess}{TimePerIteration} = \frac{33.3\ ms}{\frac{0.0372\ ms}{Iteration}} = 8951\ Iterations$$

One issue with this limitation is that when there are more features detected the probability of finding an accurate representation of the limbus is lowered. So to ensure better accuracy we try a massively parallel approach.

Since each RANSAC iteration doesn't depend on the previous iteration of RANSAC we can use a Graphical Processing Unit(GPU) to execute many iterations of RANSAC at once. To do this we take the loop and execute it once per thread using a GPU and try different threads per block to issue.

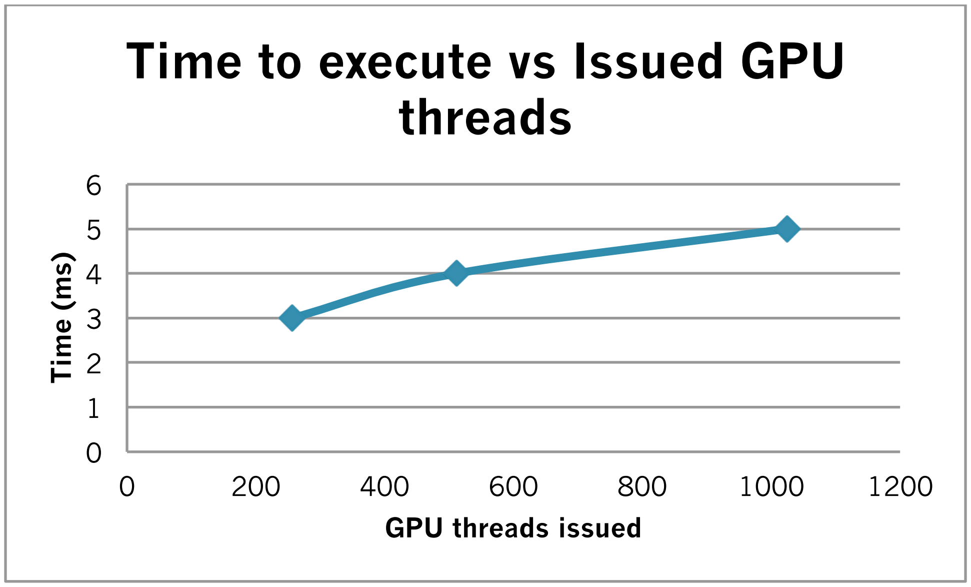


FIG XIII: Execution Time for RANSAC vs. Threads per Block (per 1 Block)

From this we find that the most efficient number of threads is 1024, which is the max number of threads we can issue on the architecture of GPU we have available.
We next try to optimize the number of blocks. These are collections of threads which are executed by a single processing unit on the GPU.

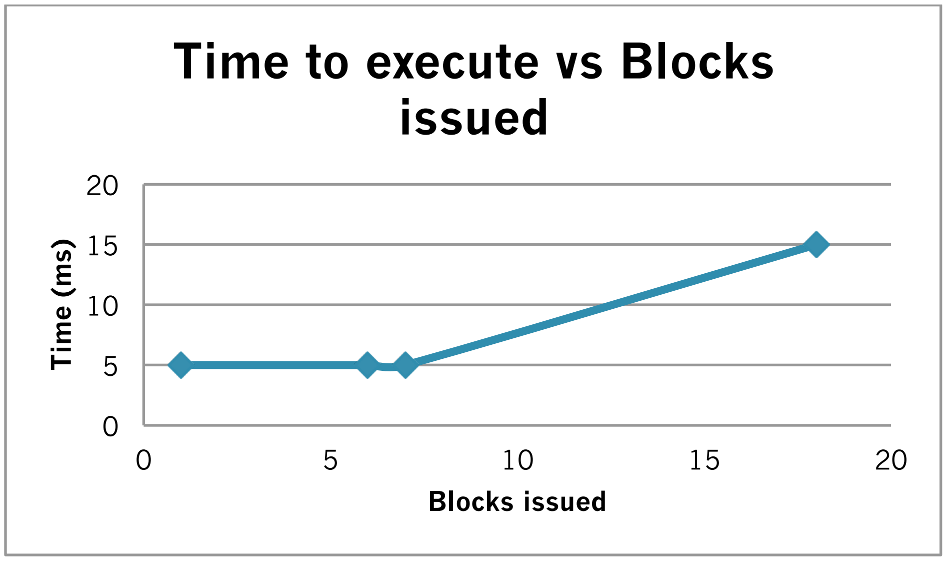


FIG XIV: Execution Time for RANSAC vs Blocks Issued

We predict that the optimal number of blocks to be issued is 7, since our GPU has 7 processing units on it.

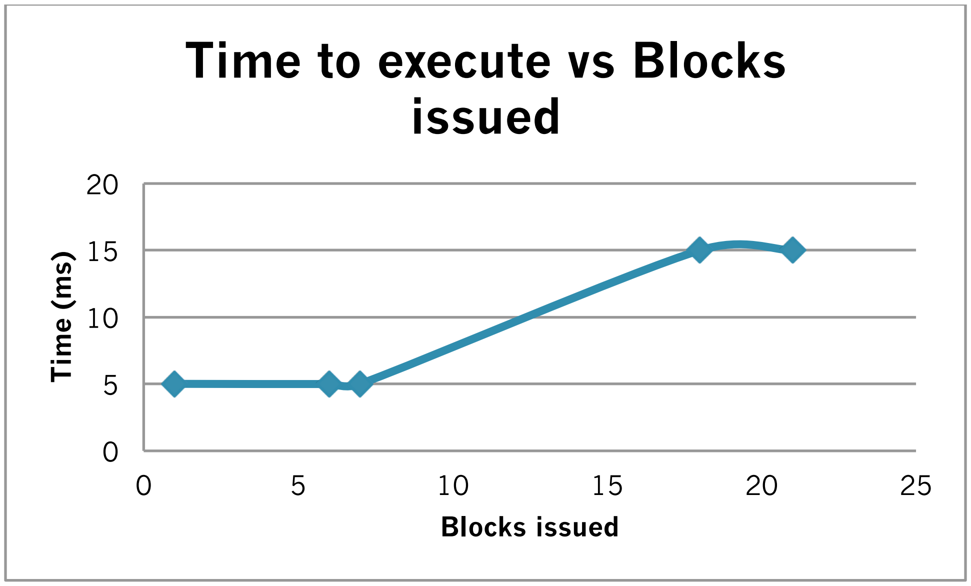


FIG XV: Execution Time per 7 Blocks

Testing shows this to be correct, as below 7 blocks results in no drop in performance, while beyond 7 blocks results in increase processing time for every 7 blocks issued. The execution time remains below our max allowable time at 21 blocks. Beyond this many blocks results in memory issues, so our optimal issue is 21 blocks of 1024 threads each. This allows us to execute 21504 iterations in 15 milliseconds.

The GPU's performance allows us to increase iterations which gives us an increased probability of detection while ensuring performance requirements.

4.2 Accuracy Testing

Accuracy tests consisted of gazing at multiple stationary points across the screen and measuring the average dispersion in the calculated point from the true location of the marker being looked at. The results are displayed below:

The average dispersion was calculated as 0.769, which is an improvement on the original Starburst algorithm as well as previous iterations of TBD's software.

Various eye and face types were tested, including varying eye colors as well as faces with make-up applied. The accuracy of ellipse fitting generally decreased in both cases. In the former, this was due to the fact that the same threshold parameters cannot be applied to the smaller differences in contrast between sclera and iris in those with lighter eyes. In the latter, this was because the addition of makeup introduced strong features that would be added as candidates for the limbus.

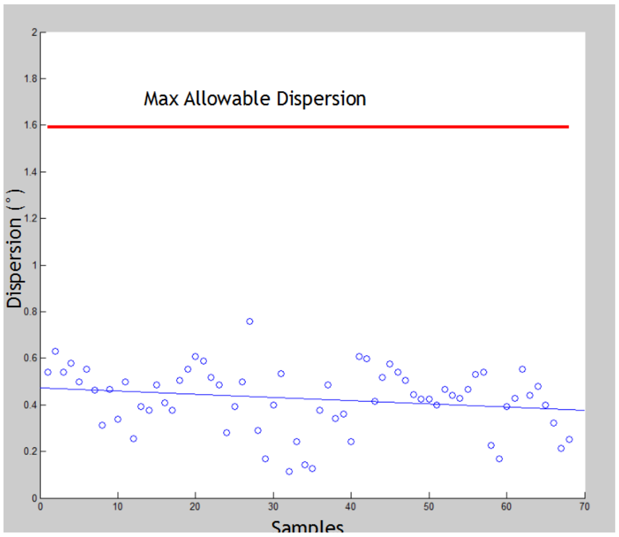


FIG XVI: Directional Accuracies Sampled

A possible solution to the former is to detect for the pupil instead of the limbus, or to view the eye-camera in colour rather than grayscale and pick pupil candidates by colour difference rather than intensity. Solving the latter is more difficult as it interferes one of the core principles behind the algorithm – assuming points of high contrast have a high likelihood of being the pupil center.

Lighting also caused significant issues. When tests were performed in areas like Vari Hall, odd lighting conditions led to situations such as shadows being cast over eyelids, which then introduced points of contrast which were taken as candidates for limbus fitting. Another issue was the dominance of reflections in certain areas. Too many reflections cause large white areas within the limbus, which cause mistaken features to be detected within the eye, reducing the probability of correct fitting.

The ranges of glyphs ranged from 1 to 3.5 inches. From user testing, we concluded the ideal size for the glyphs was approximately 2 inches. Larger sizes were not ideal as they consumed more screen space resulting in less degree of freedom in the users head motion (as the large glyphs would often disappear off the range of the camera). On the other hand, smaller glyph sizes were often not detected by the software.

Another modification, which came about due to testing, was the modification in the Kalman filter measurement. This was a result of analyzing latency and our goal to keep the delay in the cursor at a maximum of 4 frames in image processing terms. Larger delays resulted in an unnatural feeling when using the device and lack of responsiveness.

5 DEMONSTRATION & APPLICATION

To test out the TBD Eyetracker we used a game created by a third party. This game is similar to the well-known game Asteroids, but is played with a mouse. It involves asteroids falling from the top of the screen, while the user shoots the asteroids by pointing the mouse at the point of interest. This tests the several aspects of the eye-tracker including accuracy and latency. Our tests showed that the users were able to play the game well enough to accomplish all the game tasks using only their eye movements.

6 Future work

6.1 Guided Adjustment

A large source of current error comes from the adjustment of the camera relative to the eyes, which is inconsistent and varies every time it is put on. Our plan to implement a guided adjustment system. That is, when the program is run, the user will be prompted to adjust the camera until the picture received matches a predefined ideal state, such as the limbus size being ideal. This eliminates some of the randomness associated with such adjustment.

6.2 Head Mount

The future of our head mounts is constrained by cost. The head mount's physical design can be modified by increasing its visual appeal. This can be accomplished by embedding the cameras in a frame similar to eyeglasses resulting in a device that can be worn by the user and have minimal intervention from the caregiver.

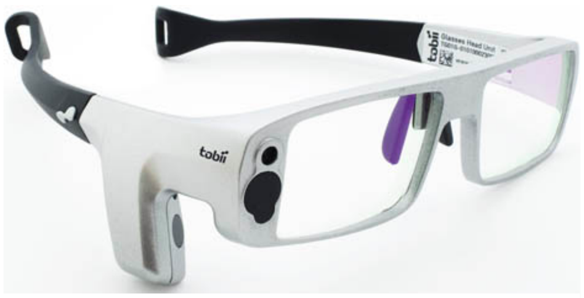


FIG XVII: Future Hardware Prototype [7]

7 BUDGET

TBD was allotted one thousand dollars for this project. There were initial estimations of using half this amount. The current estimation is at cent percent, as our previous budget included a cost one camera. The current design asks for two cameras and we have purchased four cameras to build two functioning prototypes. We are operating on budget. An additional head mount was purchased to test the next generation of our prototype but it was not used due to difficulties in implementation. An important distinction needs to be addressed here between the product cost and development cost. The cost for one of our devices is approximately $250, which includes two cameras and a head mount and Table I is the most complete and accurate representation of the development cost.

Table II: Budget

Item	Quantity	Cost (Approx.)	% of Total
Website	1	$15.00	1.5%
Camera	4	$84	33.6%
Head Mount	2	$100.00	20%
Software Licensing	1	$0	0%
Promotional/Presentation Material	1	$200.00	20%
Mounting Materials	1	$49.00	4.9%
Misc.	1	$200.00	20%
Total Expenses		**$1000.00**	**100%**
Total Revenue		**$1000.00**	**100%**
Remaining		**$0**	**0%**

8 IMPACT

Our eye-tracker will impact lives of individuals with disabilities by allowing them to use personal computers using their vision. This device is safe as its physical components are similar to headphones - with a head mount and a wire connected to the computer. The ideal conditions include indoor environments that produce minimal reflections in the user's eye. This environment does not require any special accommodations, and the users can use our device in an area they are comfortable in.

The eye-tracker is sustainable as its major components can be easily replaced and upgraded. The current prototype is approximately two hundred and fifty dollars, and majority of this cost is for the cameras. As cameras become cheaper, this head mounts can also be reduced in price, making this an economical solution.

We strongly believe this product can increase the quality of life for individuals that are unable to use computers due to physical limitations.

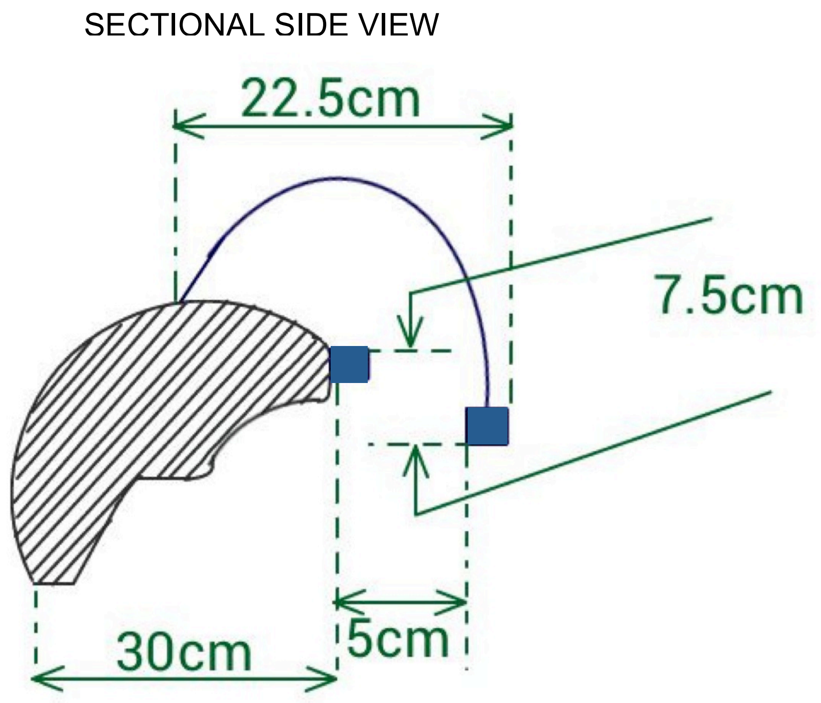
9 CONCLUSION

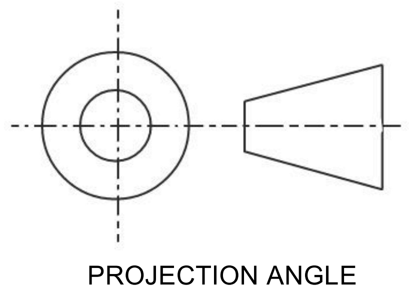
TBD is pleased with the results produced. We have developed two prototypes to test our eye-tracking solutions and proved eye detection and tracking can be used in real life application.

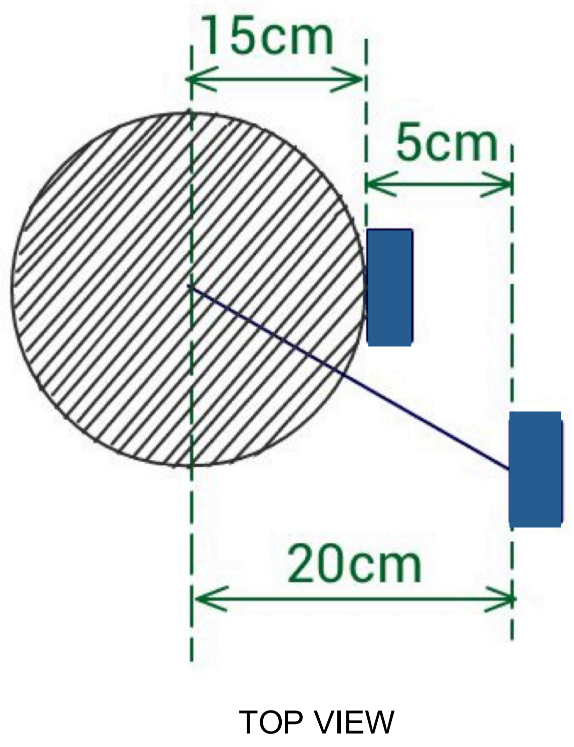
To summarize the final design report, we have demonstrated a successful project and product. The obstacles and risks have been addressed and we understand there is always room for improvements. However, we are pleased by our work as it performs at par with the products available in the market at over 25 times our cost!

This analysis of our work has indicated a healthy work group and we have delivered a product that will not only be a functional device, but also make computing more accessible.

APPENDIX A: HELMET CAMERA SPECIFICATION







This drawing is a second draft of the physical eye-tracker prototype. It was built using off-the-shelf components that are made of a combination of metals and plastics. Head mount viewed from the side and top. The cameras are in blue, helmet is black and green dashed lines indicate the approximate distance between features. The two cameras, colored blue, in this drawing are Logitech c920 mounted to the helmet (see APPENDIX B.) The helmet, colored black, used is a GIRO g9 Snow Sports Helmet Adult. The camera wires are not visible in this drawing Features mentioned include ABS Injection shell for durability, EPS Foam liner offers the best impact absorption, Fully CPSC certified and 11 Vents.

SHEET 1 OF 1

TBD GROUP 2013
GRAPHICAL COMMUNICATION
3ND PROTOTYPE

FINAL DESIGN REVIEW

DATE: APRIL 25 2013

Appendix B: CAMERA SPECIFICATION

Logitech HD Pro Webcam C920

Camera specification from the manufacturer: [8]

System Requirements

- Windows Vista®, Windows® 7 (32-bit or 64-bit) or Windows® 8

- *For HD 1080p video recording:*
  - 2.4 GHz Intel® Core 2 Duo processor
  - 2 GB RAM or more
  - Hard drive space for recorded videos
  - USB 2.0 port (USB 3.0 ready)

- *Recommended requirements for full HD 1080p and 720p video calling*:*
  - 1 Mbps upload/download for 720p
  - 2 Mbps upload/download for 1080p (Requirements for H.264 and MJPEG formats vary)
    Visit your preferred video calling provider's website for exact information on system and performance requirements.

- *For Skype® in Full HD 1080p*
  Skype 5.8 for Windows*

Warranty Information

- 2-year limited hardware warranty

Package Contents

- Webcam with 6-foot cable
- User documentation

Part Number

- PN 960-000764

Technical Specifications

- Full HD 1080p video calling (up to 1920 x 1080 pixels) with the latest version of Skype for Windows*
- 720p HD video calling (up to 1280 x 720 pixels) with supported clients
- Full HD video recording (up to 1920 x 1080 pixels) with a recommended system**
- Logitech Fluid Crystal™ Technology
- H.264 video compression*
- Carl Zeiss® lens with 20-step autofocus
- Built-in dual stereo mics with automatic noise reduction
- Automatic low-light correction
- Hi-Speed USB 2.0 certified (USB 3.0 ready)
- Tripod-ready universal clip fits laptops, LCD or CRT monitors

Logitech webcam software:***

- Video recording: Up to Full HD 1080p video capture**
- Photo capture: Up to 15 megapixels (software enhanced)
- 1-click Facebook®, Twitter™ and YouTube™ HD upload (registration required)

**Please download the latest version of Skype, Skype 5.7 Beta for Windows, which offers 1080p HD video calling.*
*** H.264 recording requires installation of QuickTime®.*
**** Requires installation of included softwar*

REFERENCES

[1] "Disability Statistics Canada." *Disabled World News and Disability Information*. N.p., n.d. Web. 27 Mar. 2013. <http://www.disabled-world.com/disability/statistics/disability-statistics-canada.php>.

[2] Li, Dongheng , and Derrick Parkhurst. "Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches." *IEEE Xplore* 10.1109/CVPR.2005.531 (2005): n. pag. *IEEE Xplore*. Web. 27 Mar. 2013.

[3] Robert, Fisher. "The RANSAC (Random Sample Consensus) Algorithm."Informatics Homepages Server. N.p., n.d. Web. 5 May 2013. <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FISHER/RANSAC/>.

[4] "AForge.NET :: Glyphs' recognition." AForge.NET :: Computer Vision, Artificial Intelligence, Robotics. N.p., n.d. Web. 29 Mar. 2013. <http://www.aforgenet.com/articles/glyph_recognition/>.

[5] "OpenCV | OpenCV." *OpenCV | OpenCV*. N.p., n.d. Web. 5 May 2013. <http://opencv.org/>.

[6] " ArUco: a minimal library for Augmented Reality applications based on OpenCv | Aplicaciones de la Visión Artificial." *Universidad de Córdoba*. N.p., n.d. Web. 5 May 2013. <http://www.uco.es/investiga/grupos/ava/node/26>.

[7] "Tobii Glasses | Tobii Eye Tracker IS | from Acuity ETS." *Usability & User Experience | Tobii Eye Tracking from Acuity ETS*. N.p., n.d. Web. 29 Mar. 2013. <http://www.acuity-ets.com/products_glasses.htm>.

[8] "Logitech® HD Pro Webcam C920." Logitech - Get Immersed in the Digital World!. N.p., n.d. Web. 29 Mar. 2013. <http://www.logitech.com/en-us/product/hd-pro-webcam-c920?crid=34>.

References not cited but used:

Szeliski, Richard Springer. "Computer Visison." Computer Vision Algorithms and Applications. York University. Toronto, ON. FALL 2012. Class lectures.

Drewes, Heiko. *Eye gaze tracking for human computer interaction*. Munich: Disseration, Ludwig Maximilians University, 2010. Print.

Richard Hartley, Andrew Zisserman, *Multiple View Geometry in Computer Vision* Cambridge, UK; New York : Cambridge University Press, 2000

CONTRIBUTION SHEET

TEAM MEMBERS & ROLES

Josh Sideris
jsideris@tbdgroup.ca
Chief Research Officer

Ragavan Thurairatnam
ragavan@tbdgroup.ca
Chief Technical Officer

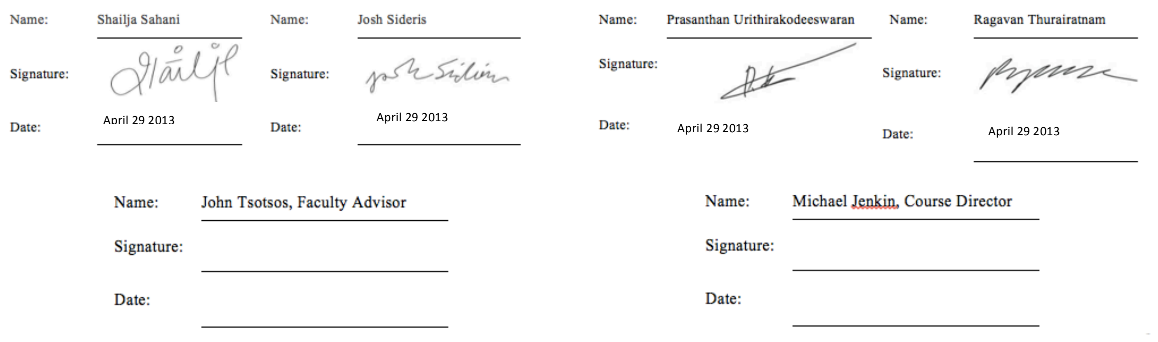
Prasanthan Urithirakodeeswaran
pkodees@tdbgroup.ca
Chief Development Officer

Shailja Sahani
shelly@tbdgroup.ca
Chief Business Office

MEMBER WORK BREAKDOWN

Table III: Contributions

Title	Josh S.	Ragavan T.	Prasa U.	Shailja S.	Total
Project Management	50%	0%	0%	50%	100%
Technical Reports	10%	15%	25%	50%	100%
Vision Development	30%	70%	0%	0%	100%
UI Development	60%	0%	40%	0%	100%
Interface Development	60%	0%	40%	0%	100%
Weekly Team Meetings	25%	25%	25%	25%	100%
Market Research	10%	10%	20%	60%	100%
Software Architecture	0%	0%	100%	0%	100%
Hardware Selection	0%	50%	0%	50%	100%
Finance	0%	0%	0%	100%	100%
Testing	25%	25%	25%	25%	100%
Demonstration	20%	20%	20%	40%	100%
Success Measurement	25%	25%	25%	25%	100%



Name: Shailja Sahani
Signature: [illegible]
Date: April 29 2013

Name: Josh Sideris
Signature: [illegible]
Date: April 29 2013

Name: Prasanthan Urithirakodeeswaran
Signature: [illegible]
Date: April 29 2013

Name: Ragavan Thurairatnam
Signature: [illegible]
Date: April 29 2013

Name: John Tsotsos, Faculty Advisor
Signature:
Date:

Name: Michael Jenkin, Course Director
Signature:
Date: