Eye Gestures Recognition:
A Mechanism for Hands-Free Computer Control

Issa Ashwash, Willie Hu & Garrett Marcotte

Introduction

The notion that useful conclusions could be drawn from information about the movement of a person's eyes has been around for well over a century. For a sighted person, the primary means of interacting with the world is through visual input, so it is natural to assume that information about what a person is looking at in any given moment would be instrumental in determining how that person is interacting with the world. In the field of human-computer interface design, knowledge about such interactions can be critical to designing a powerful, intuitive and ultimately helpful user interface.

Eye tracking began in the late 1800s with mechanical devices that tracked light reflection patterns or even materials directly embedded in the cornea. With the growth of photography and video recording technology, far more reliable and less invasive means were developed to simply observe a user's eye motions during long periods of activity. These recordings would then be analyzed manually, often on a painstaking frame-by-frame basis, generating mountains of data to analyze. The task was daunting and even in a perfectly performed experiment, the data could very easily defy all attempts at understanding. The growth of computing technology eventually eased the data analysis task to the point of feasibility, and eye-tracking experiments became more popular, and devices for incorporating eye gaze data into psychological and medial studies began to appear. It is only recently, however, that computing power and video recording capability have become inexpensive and powerful enough to cross the gap into the demanding, real-time world of interface design. It is this frontier of the eye-tracking field that our project seeks to explore. (For a detailed history of eye-tracking research, see [3]).

Eye tracking devices have historically fallen into two categories. The first is passive, and focused on detecting the gaze of the user relative to the rest of the world and in particular what elements of the visible field are currently being focused upon. The second is more active, and considers the eye not as simply a means of observation, but a means of control as well. We think that a device capable of identifying deliberate movements of the eye area (pupils, eyelids and eyebrows), we can provide a new means of interaction that could replace or complement more standard interfaces.

Similar Work

Eye tracking is somewhat unusual as a field, in that it has been the subject of intense research for decades, and yet never reached the level of accuracy, usability and costefficiency to become a widespread means of human-computer interaction. Countless times (detailed in [3]), some new development in eye tracking has been heralded as the spark to start an HCI revolution, and each time that revolution inevitably failed to materialize. These failures led researchers to explore new directions and ideas, to the point that there are currently three major techniques in use that we considered for this project.

First is a biological measurement technique called an Electro-Oculogram (EOG). The device consists of pairs of electrodes attached around the eye (often either right and left or top and bottom). Inside of the eye is an area called the retina, which carries an electric charge gradient. When eye rotates, this charge gradient produces a potential difference between opposite sides of the eye, which can be detected by the electrons. Unfortunately, this signal is easily corrupted and tends to drift, making accurate detection difficult. Patmore and Knapp ([5]) have successfully created an eye motion detector based on this measurement that uses fuzzy logic techniques to remove this drift and have used it to control the cursor on a computer screen. Because of the signal inaccuracy and physical intrusiveness, we chose not to use this method.

Second is the "Dark Pupil/Light Pupil" technique using infrared light. Under infrared illumination, the pupil becomes very white, almost the exact opposite of its visual-spectrum appearance. By capturing both the dark and light pupil images, the high contrast (which is mostly localized to the pupil) can be used via image subtraction to evaluate the pupil location with very high accuracy. (See [4] and [6] for details on the technique, and a working implementation). Because we wanted to leverage more eye data than just pupil location, and because we wanted to use widely available equipment as much as possible, we chose not to use this method.

The final method uses plain visible-light cameras and computer-vision techniques to extract details about the position of various interesting features. The growth of the computer vision field in the last ten to fifteen years has led to a multitude of techniques that are capable of performing such analysis. See [4] for a general discussion. For examples, see the USB Webcam Blink Detector by Chau and Betke ([2]), the Starburst Algorithm by Li and Parkhurst ([9]), and Savas' TrackEye software ([7]). One benefit of this method is that it doesn't rely on characteristics that are extremely specific to the eye (e.g. retinal charge gradients or infrared pupil reflection), and can be tailored to other features of more complex interactions (see, for example, the Camera Mouse by Betke *et al.,* [1]). Because of the variety of options, the ability to use ordinary cameras, and some group member experience in computer vision, we decided to use these techniques for our project.

Feature Tracking

The tracking system is the first step in the eye-tracking device and is the most directly in contact with the user. Tracking processes all of the data from the camera input(s), presents the calibration and parameter-tweaking interfaces to the user, and performs computer vision-related algorithms to determine the location of the user's pupils and eyebrows. From a user-interface perspective, the operation of the device is as follows:

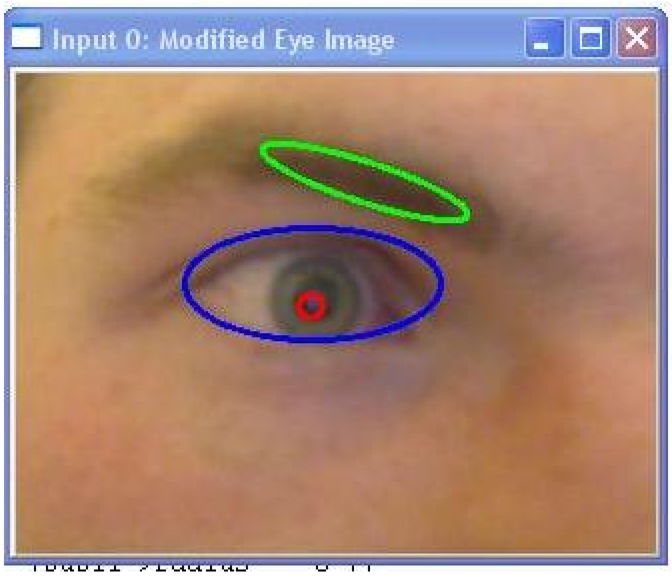
- When the program begins, the user is prompted to position the camera and headgear properly so that the full regions of both eyes are clearly visible and unobstructed in the video.
- Then the user is prompted to select the appropriate eye regions from a video snapshot. The selection of the eye region is very important because it allows us to

significantly narrow our search space, ensure that only eye-related features appear on the image, and cut down on external interference with the tracking process.

- With the regions selected, detailed tracking windows appear and the user is prompted to configure (via a set of trackbars) the parameters that determine tracking. This is certainly the clunkiest part of the system, and the part that would benefit greatly from improvement in a more polished version of the device. Specifically, the trackbars control the thresholding level applied to the input images. The user does not need to know any technical details since the best-guess estimate of pupil and eyebrow position are superimposed on the camera input, and the user need only slide the various trackbars until the estimates are steady and follow his or her movements.
- After the parameters are set, calibration mode begins. In this stage, the user is prompted conduct a few eye and eyebrow movements while the program trains. The specific required actions are looking left, right, up and down, and raising the eyebrows. Once calibration is complete, the user is ready to begin using the device. The device can be recalibrated or disabled at any time with a simple keystroke.

The tracking algorithms are both based heavily on the contrasts and contours contained in the image of the eye. The pupil recognition algorithm is borrowed from Zafer Savas' TrackEye software ([7]). The image is first thresholded and then run through a Canny edge detector. Then contours are found and the contours are filtered to extract the largest, most circular closed contour available. In a sufficiently controlled image with proper parameters, this contour will be the pupil. The eyebrow recognition follows a similar process, and was custom written by our group. Very few existing eye trackers that we could find dealt at all with eyebrows, and we think eyebrow raise/lower recognition could provide useful control motions. The eyebrow recognition algorithm uses thresholding and contour finding, and then looks for the contour highest up in the image that is sufficiently large and oval-shaped. For this reason, it is important that the eye region be cropped short enough so that it doesn't contain other long/oval-shaped contours higher up (such as hair or the underside of the hat).

Fig 1: Tracking software showing it's best estimates of the pupil, eyebrow and eye locations.



Gesture Recognition
Gesture Recognition's design was really a result of the limitations in Tracking. If tracking could be assumed to be perfect, Recognition would naturally be very trivial - similar to any system that tries to utilize real time, world information. Recognitions stated purpose is to ignore blinking, recognize winking, recognize eyebrow raising, recognize both eyes being closed deliberately, and recognizing deliberate eye movements left, right, up and down. To achieve these operations, Recognition's goals therefore included: using temporal changes in the gaze to ignore outliers, filtering the pupil location over time to minimize error, and using assumptions about the image and the effectiveness of tracking to improve recognition. Initially we will discuss these goals then to establishing the parameters for recognition.

Temporal Outliers
The camera operates at thirty hertz - quickly enough so that most actions occur over a series of frames. For example, natural blinking occurs over the course of 2 or 3 frames. We use that same time frame for a person to input a directional input (they must look upward for 3 frames).

Time Filtering
The pupil location and its radius are filtered over time with a Moving Average Filter. Tuning this over time revealed that storing three frames was sufficiently accurate. This is especially helpful for the radius, which can vary by two or three factors in bad tracking areas.
This does not aid the tracking of eye position precisely. Rather, this is geared for when the user is trying to send a directional input: the added filtering improves the rate at which the software recognizes directional input, and helps minimize outliers skewing directional input.
The recognition software was also set to be able to be recalibrated at any point, as well as recieve extra calibration data at any point and integrate it into the running constants.

Assumptions Made
Since we are looking at both eyes simultaneously, we have an excess of data which can be whittled down to ensure the minimization of false positives and false negatives.
One of our primary assumptions is that if a pupil is not recognized, the eye is considered to be blinking.
Based on our own observations it was clear that the tracking software was better able to correctly recognize the pupil when the eye was looking inwards towards the center of the face (towards the other eye as opposed the nearest ear). Therefore, if either eye is seen looking inwards, that is taken as the correct gaze and the other is ignored. Furthermore, the tracking software tended to lose the pupil very easily if the eye was looking downwards, so the recognition software allows for gaps (by considering a blink during a series of downward eye actions to increase the count).

With those features implemented, recognizing specific actions became relatively simple. The main recognizer loop was a function which updated after every image frame: the tracking software would call the Recognizer Class object's update loop, passing it the collected data for each eye. The implementation used a finite state machine approach to go through every fame, adjusting the counts above to implement the temporal filtering and adding the necessary variables and adjusting the moving average filter. The assumptions were implemented as part of the state machine. Variables that stored counts were used as temporal filters - actions had to continue for a prespecified number of frames before it would activate.

Once an action is determined, a keyboard event is sent, corresponding to the expectations used in testing. The eyebrow moving up would be sent as a 'w', right would be a 'd', eye winks were 'q' or 'e', a blink would b 'r' and an eyebrow raise would be 'f'.

Hardware

The physical device consists of a baseball hat with a USB webcam attached to the brim and mounted in front of the face. Our testing used Microsoft LifeCam VX-5000, selected because it was the cheapest webcam from a reputable vendor in Princeton Township. The camera is adjustable: it can be turned left to right on its mount with about 120 degrees of freedom, as well as angled up and down with about 45 degrees of freedom, however it cannot be brought towards or away from the face. Adjusting distance is not necessary, as our tracking software is not dependent on the size of the image. The webcam is plugged directly into the host computer.
The current version is a prototype; future production versions would employ a purpose-built hat with much smaller cameras directly integrated into the construction (such as Volvox USB webcam, or smaller chipsets similar to camera phones). These modification will lead to a much sleeker and more durable device. The cameras would also have a wireless interface so as to impede the movements of the user as little as possible.

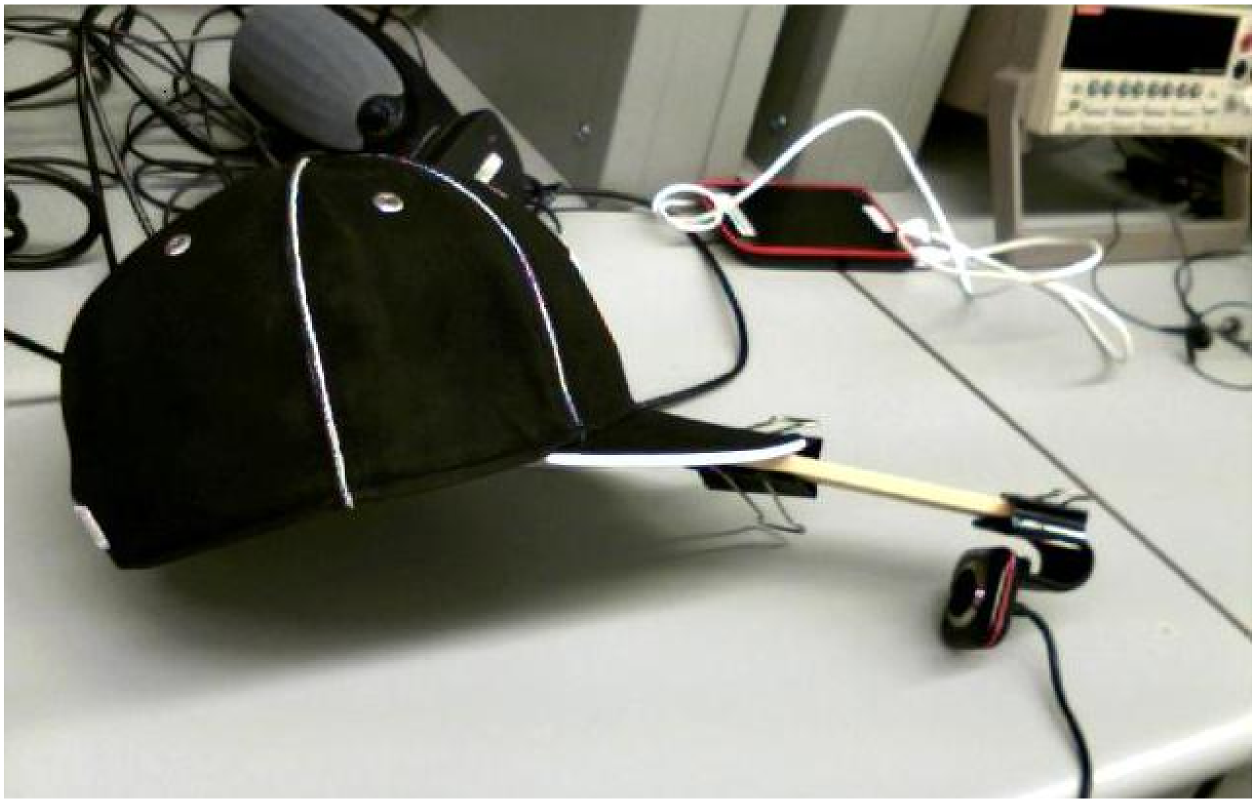


Fig 2. The Camera Hardware and Wearable Device

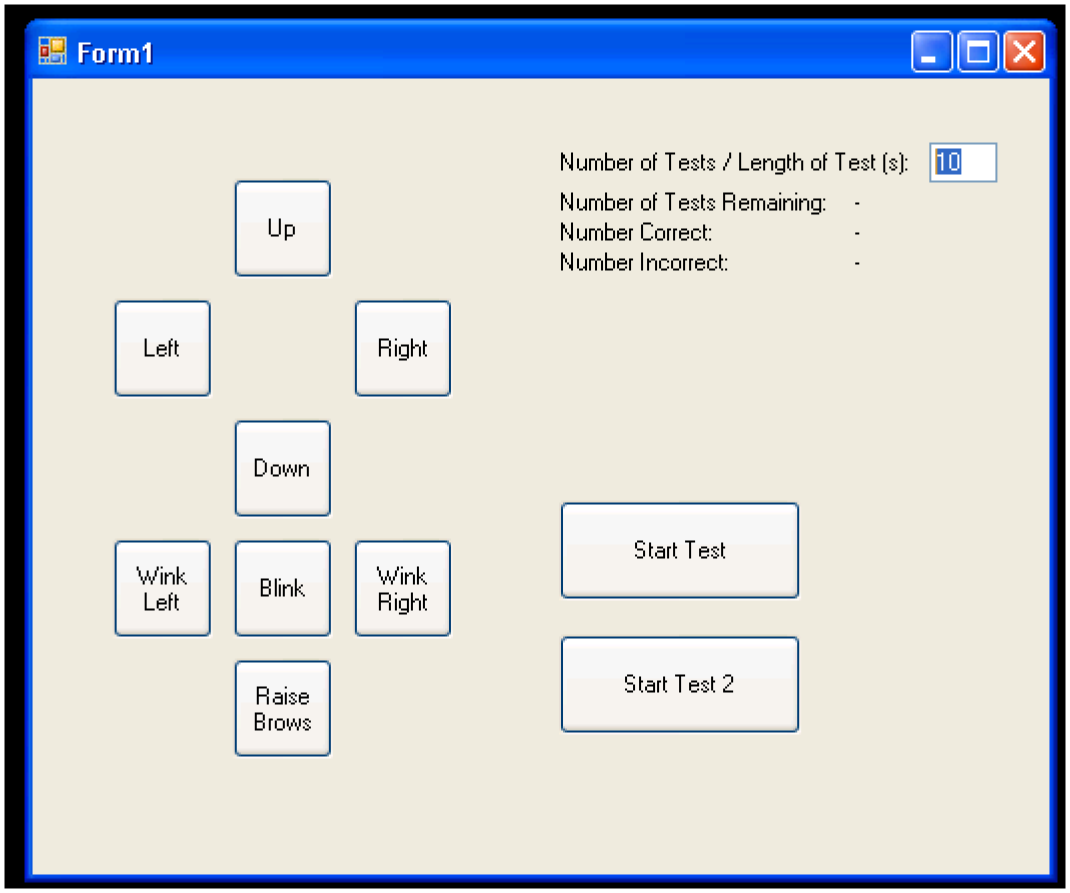
Testing

We constructed two separate experiments to test the usability of our Eye Gesture Tracker. Both can be run from a simple Tester app designed for this purpose. All testing applications communicate with the Eye Gesture Tracker driver via keystrokes, as described above.

The first experiment is designed to test the recognition accuracy of the Eye Gesture Tracker. It consists of the test application randomly prompting the test subject to execute one of the eight Eye Gesture signals (look up, look left, look down, look right, wink left, wink right, blink, or raise brows) for a pre-determined number of trials. Each prompt, its response, and the time required to perform the response are all recorded.

The second experiment is designed to test the false positive rate of the Eye Gesture Tracker. It involves asking each test subject to simply sit in front of the computer wearing the device while deliberately avoiding to make any one of the eight Eye Gesture signals. Any Eye Gesture signals detected are recorded.

Each subject was given an explanation of how the Eye Gesture Tracker functioned and given a few minutes to experiment with it. Then, the first experiment was administered for a total of 4 trials (each of the 8 actions were tested 4 times each) followed by the second experiment for a length of 2 minutes. The results are discussed below.



Number of Tests / Length of Test (s): 10
Number of Tests Remaining: -
Number Correct: -
Number Incorrect: -

The Test application.

Development Process

Originally, the physical device consists of a baseball cap with two identical USB webcams attached to the brim, one to each side. This mounting method gives each webcam a clear, unobstructed view of each wearer's eye. Both USB webcams are plugged directly into the host computer.
This was due to our initial assumption that we would be able to track the eye significantly more effectively, and therefore having dedicated camera on each eye would not only be useful, but the increased image resolution would improve the tracking substantially. As it turns out, detecting the bounding box of the eye was significantly more error prone than originally anticipated. Thus, we scrapped that portion of the design and switched to a monocular approach. In this method, we require a calibration stage for the user. While this is an added bother for the user, at this point in testing we require manual tuning of the image at startup for optimal detection anyway, so the calibration portion is not especially cumbersome in contrast.

Results

Due to the technical difficulties listed below in the Known Limitations section, we did not obtain any meaningful results from test subjects not affiliated with the development of this project.

A small practicality - the hat used for the experiment was a fitted cap 7 and 3/4" size. It was a large hard and slid around easily. A cap that can adjust its size would be more comfortable for the user and would improve image stabilization. However, in general, a cap mounted design proved as an effective mount for the camera.

Known Limitations

One major issue with the cameras currently being used is that low contrast among eye/ facial features renders the tracking system un-operable. In other words, individuals with brown/dark pupils and/or eyebrows closely matched to their skin tone are hard to track, due to the small differences in the contrast between these features. Higher quality cameras with greater contrast ratios may remedy this problem.

Another camera related problem is that of the camera picture/focus obtained. Because the device consists of a one-size-fits-all baseball cap, there are obviously differences between the quality of the images obtained between different users. The Eye Gesture Tracker requires a clear, unobstructed view of each individual's eye in order to work best. However, sometimes this is hard to obtain based on the construction of the device and each individual's anatomy.

Ideas for Improvement

As mentioned above in the hardware section, this device could certainly benefit from better construction. The Eye Gesture Tracker system could also benefit much from improved software. The start-up process is currently a very tedious process requiring very careful calibration. For the Eye Gesture Tracker to be commercially successful, it would need a much simplified and very streamlined start-up procedure.

The Eye Gesture Tracker is a fairly basic input device, providing only eight mappings. Thus, in its current state, it is not very well suited to be used on its own. To increase the number of mappings the system can provide, the Gesture Tracker could be extended to support sequences of signals, for example looking up and then down quickly could be recognized as a signal different from looking up and then down separately. One of the authors envisions playing Street Fighter with this interface, for example.

References

[1] M. Betke, J. Gips and P. Fleming. "The camera mouse: Visual tracking of body features to
provide computer access for people with severe disabliites." *IEEE Transactions on Neural*
Systems and Rehabilitation Engineering, 10:1, pages 1-10, March 2002.
http://www.cs.bu.edu/fac/betke/papers/betke-gips-fleming-nsre02.pdf

[2] M. Chau and M. Betke. "Real time eye tracking and blink detection with USB cameras."
Boston University Computer Science Technical Report No. 2005-12, May 2005.
http://www.cs.bu.edu/techreports/pdf/2005-012-blink-detection.pdf

[3] R.J.K. Jacob and K.S. Karn. “Eye Tracking in Human-Computer Interaction and Usability
Research: Ready to Deliver the Promises (Section Commentary),” in *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, ed. by J. Hyona, R. Radach, and H. Deubel, pp. 573-605, Amsterdam, Elsevier Science (2003).
http://www.cs.tufts.edu/~jacob/papers/ecem.pdf

[4] C. Morimoto and M. Mimica. "Eye gaze tracking techniques for interactive applications."
Computer Vision and Image Understanding 98", pages 4-24, 2005.

[5] D. W. Patmore and R.B Knapp. "Towards an EOG-based eye tracker for computer control."
In *Proceedings of the Third international ACM Conference on Assistive Technologies* (Marina del Rey, California, United States, April 15 - 17, 1998). Assets '98. ACM, New York, NY, 197-203. http://doi.acm.org/10.1145/274497.274533

[6] Z. Zhu and Q. Ji. "Robust real-time eye detection and tracking under variable lighting conditions and various face orientations." *Computer Vision and Understanding 98"*, pages
124-154, 2005.

http://www.ecse.rpi.edu/~cvrl/zhiwei/html/papers/cviueyetracking.pdf

[7] Z. Savas, "TrackEye: Real time tracking of human eyes using a webcam."
http://www.codeproject.com/KB/cpp/TrackEye.aspx

[8] OpenCV Library.
http://sourceforge.net/projects/opencvlibrary/

[9] D. Li and D. Parkhurst. "Open-source software for real-time visible-spectrum eye tracking." *Second Conference on Communication by Gaze Interaction*, pages 1-3, 2006.
http://thirtysixthspan.com/openEyes/li_parkhurst06.pdf

[10] G. Bradski and A Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library.*
United States, O'Reilly Media. September 2008.