iSocial Open Source Eyetracking Investigation

Gordon Graber

September 7, 2009

This report is the product of an investigation into open source eyetracking systems for research in conjunction with the University of Missouri iSocial project - http://isocial.missouri.edu/.

Turnkey eyetracking systems are expensive - the lowest cost systems commercially available that I have been able to find start at around $8,000 (A Comparison of Eye Tracking Tools in Usability Testing: http://tc.eserver.org/29563.html; Arrington Research: http://www.arringtonresearch.com/prices.html. I was not able verify prices, features or the viability of these lower cost commercial systems.) Because of the costs, finding a usable open source system may be desirable.

While using open source eyetracking systems for research certainly seem to be possible now that I have completed this investigation, much of the time spent in my research was consumed by reconciling conflicting information from users and developers, and seeking out answers to problems and issues with the software and hardware I tested.

Software and Hardware Examined

Open Source Eyetracking Software

A Web search of eyetracking system information revealed a site dedicated to open source eyetracking systems - COGAIN: Communication by Gaze Interaction (http://www.cogain.org). COGAIN is a consortium of European organizations primarily interested in eyetracking as a means of input and control for research and use by people with muscular disabilities.

One page at the COGAIN Web site is dedicated to open source eyetracking systems. Two eyetracking systems were identified, **ITU-GazeTracker** (http://www.gazegroup.org/) and **TrackEye** (http://www.codeproject.com/KB/cpp/TrackEye.aspx). All the other systems listed at COGAIN were not available as downloadable products.

The specific choice of computer platform and camera was determined by the eyetracking software.

Choosing ITU-GazeTracker over TrackEye

ITU-GazeTracker and TrackEye accomplish eyetracking and data collection using different methods. The following is a comparison of the features of each that ultimately led me to choose ITU-GazeTracker. The choice was itself a process of trying the software, attempting to overcome problems, digging through the discussion forums, contacting the developers, and traveling down a road of diminishing options.

- Both ITU-GazeTracker and TrackEye are under various stages of development, as I soon discovered when downloaded both and attempted to run them using a standard Logitech Web camera.
- Both have a small user base – 10 to 20 apparently active users, as evidenced by discussion board activity available at their respective Web sites.
- ITU-GazeTracker has the backing of several people at the IT University of Copenhagen (http://www.gazegroup.org/people) and lists of number of research endeavors. TrackEye appears to be the work of a single author.
- Both systems provide minimal documentation. Of the two, ITU-GazeTracker's documentation was more comprehensive.
- ITU-GazeTracker (potentially) outputs eyetracking data to a log file as well as to a UDP data stream. TrackEye only outputs to UDP. (*UDP is a real-time TCP/IP streaming protocol useful for device control.*)
- ITU-GazeTracker provides a tracking calibration process. TrackEye has none that I could find. (*Calibration ensures that eyetracking coordinates are fixed to the computer display's frame of reference.*) It is not clear how TrackEye effects calibration.
- TrackEye uses a stationary camera placed near the computer display and provides compensation for head movement by tracking head as well as eye position. ITU-GazeTracker can be configured for either a head mounted camera or a stationary camera. Using a head mounted camera, head position must remain fixed for the calibration to remain valid throughout data collection. This would require some type of headrest to ensure minimum head motion. In the stationary camera configuration, ITU-GazeTracker requires an infrared light source pointed at the subject to use glint tracking. Tracking glints (reflections) off the eye, which remain in a fixed position irrespective of eye motion, as well as eye movement allows ITU-GazeTracker to compensate tracking coordinates for head motion.
- ITU-GazeTracker requires an infrared capable camera. TrackEye uses visible light cameras. Eyetracking systems using infrared light have an advantage because the images they capture are less sensitive to environmental lighting variations.
- ITU-GazeTracker listed out 3 cameras that were known to be usable, with infrared capability, 2 of which are available for purchase in North America. TrackEye could potentially use any digital camera. (*Virtually all digital cameras have infrared capability, but have an infrared filter covering the lens. Converting a camera to allow it to capture infrared light is simply a matter of removing the filter, as I described below.*)
- ITU-GazeTracker's Web site is more comprehensive. TrackEye's Web site consists of one page.

- Both systems require Windows XP computers with Intel Dual Core processors and 2GB RAM.
- ITU-GazeTracker's user interface is friendlier than that of TrackEye

Analysis Software

After searching the Web for open source gaze analysis software, I found OGAMA (Open Gaze And Mouse Analysis) (http://didaktik.physik.fu-berlin.de/projekte/ogama/), to be the only full featured analysis system that didn't require compilation (existed as an executable download) and offered the promise of importing gaze data from eyetracking systems. While it is certainly possible to use MatLab or some other data analysis system, OGAMA is the only open source alternative that approximates if not exceeds the features of turnkey systems.

OGAMA processes eyetracking data with static images only. Another analysis package, iComponent (http://www.cs.uta.fi/~oleg/icomp.html), apparently does process eyetracking data over moving images, but does not allow data imported – iComponent uses a plug-ins architecture to receive data directly from commercial eyetracking systems.

OGAMA is maintained and supported by single author in Germany. OGAMA has no manual and no guide other than a slim FAQ. Support is supplied through direct contact with the author.

Summary of Challenges and Limitations Encountered

Microsoft LifeCam VX-1000 Web camera & driver software

Have to remove infrared filter and install visible light filter. Low resolution requires mounting within several centimeters of the eye. Doesn't appear to produce stable images despite fixed settings.

ITU-GazeTracker

Current build does not implement some features (data logging) delineated in the documentation. Requires rebuilding source code. Occasionally locks up at the end of the calibration process – I did not have time to follow through with this issue.

OGAMA - Open Gaze And Mouse Analysis

No documentation – use requires contact with the developer. Processes eyetracking data used in conjunction with static images only. OGAMA did not initially import ITU-GazeTracker data, due to a bug found by the developer.

Open source software documentation and support

Documentation for the open source software used was slim or none. Discussion forums are available, but small user base limits resolving problems quickly. Developers were responsive and contacting them is essential.

Problems and Issues with Open Source Systems

This section is a recounting of the sequence of events that led my decisions about software and hardware. I believe this account is useful in understanding the decisions I made and paths I chose to take in getting an eyetracking system running.

After trying to work with both ITU-GazeTracker and TrackEye I choose the former, but it wasn't an easy decision. From the start, my experience with open source eyetracking was difficult, to say the least. Upon install, ITU-GazeTracker would not work with the Logitech Web camera I had available – no image was visible, and it crashed often. TrackEye ran out of the box and displayed images and visual tracking indicators, but could only output data through UDP. Trying to gather eyetracking data via UDP would require another piece of software to collect the UDP stream from a TCP/IP port that had to run concurrently with the eyetracking software. This would add another layer of complexity to an eyetracking system to be used for research that has no need for real-time device control. This factor pushed me toward ITU-GazeTracker.

From the discussion forum, it appeared people were using ITU-GazeTracker successfully, and I reasoned that acquiring one of the recommended cameras would resolve the issues with ITU-GazeTracker displaying images and crashing.

Selecting a Camera and Modification

Having decided to proceed with ITU-GazeTracker, the documentation recommended two cameras available for purchase in the United States: a Microsoft LifeCam VX-1000 – a light weight, low resolution (640 x 480) Web camera for around $30; and Sony HDR-HC5 digital video camera - a high resolution (1080i HDTV) camera with a zoom lens and a configurable infrared feature, for around $1,500.

Due to the price, we choose the LifeCam VX-1000 for this investigation. To prepare it for use with ITU-GazeTracker, the LifeCam needed to have the infrared filter removed and a visible light filter installed. The procedure involves popping open the camera's enclosure, prying the infrared filter off the lens assembly and replacing it with a small square of developed 35mm negative film to block visible light. A tutorial detailing the procedure is available at http://www.free-track.net/english/hardware/filter_removal/microsoft_lifecam_vx1000.php

Using ITU-GazeTracker with the LifeCam VX-1000

The new camera did not solve the problem with ITU-Gaze tracker - the application still did not show images captured by the camera, and would crash often. After several hours trying to resolve the issue by searching the discussion forums and posting inquiries; contacting the software's author, Xavier, in Denmark; and reinstalling the camera drivers and ITU-GazeTracker. By chance, I discovered that the drivers supplied on the CD that came with the newly purchased camera were not the most recent available. I downloaded the new drivers from Microsoft and the problem was resolved. ITU-GazeTracker now displayed images, and I was ready to try to calibrate and record some data.

The LifeCam VX-1000 low resolution dictated that it be used with ITU-GazeTracker in a head-mount configuration. I was able to forego building head mountable rigging by attaching the camera to a large clip, and hang it from a microphone stand we have in the lab. This allowed the placement of the camera close enough to the eye to be useful.

In the head-mount configuration, the users head needs to remain motionless. This was achieved by resting the subject's chin on the back of a tall chair positioned between the subject and the computer display.

One problem I started to notice as I worked with the LifeCam VX-1000 is an apparent settings drift over time. ITU-GazeTracker relies on the LifeCam drivers to supply it with video. The drivers provide a number of adjustable values, from overall gamma, brightness, contrast, to automatic backlighting compensation. All of these are accessible through a separate LifeCam utility, installed with the drivers. Opening the utility successively revealed slightly lighter or darker video image, even though the display of the settings options did not change. This seeming drift in settings causes ITU-GazeTracker problems in determining eye position. ITU-GazeTracker provides a threshold setting to help it locate the pupil for a given image brightness, which it tracks. ITU-GazeTracker bases this threshold setting on the images supplied by the LifeCam drivers. Any change in the brightness of the supplied image will invalidate the threshold setting. ITU-GazeTracker would, after a few minuets, no longer be able to detect the pupil. By adjusting the LifeCam settings, I have been able to minimize this problem but not eliminate it, and I don't have a firm explanation of why this settings drift seems to be happening, or what setting it might be attributed to. It seems like it is related to LifeCam's automatic backlighting, but I have that setting turned that off. It may be an artifact of Microsoft's LifeCam drivers, or due to the inexpensive hardware used in the camera.

Data Recording

Once I was able to use the LifeCam and effect the ITU-GazeTracker calibration process I started recording, but to my dismay no log file could be found. I double-checked the ITU-GazeTracker settings where there is a checkbox to turn on output to a log file, and the name and path to the log file. None of these options seemed to work. In the settings field indicating output to a log file, I checked and rechecked, recorded and re-recorded, all with no results. I contacted the software author, Xavier. While waiting for his reply – correspondence often took more than a day due to his location in Denmark – I noticed a post in the discussion forum from Xavier's partner, Martin, who stated that this feature of ITU-GazeTracker was not implemented, and one had to use a UDP capture utility to get data from the system. Avoiding UDP was the main reason I had chosen ITU-

GazeTracker over TrackEye and now it was looking like I would have to deal with UDP after all.

I then located a shareware UDP capture utility that could be used on a trial basis, ComCap, which allowed grabbing local UDP streams and saving them as text files (http://www.magsys.co.uk/comcap/). ComCap needs the TCP/IP port that ITU-GazeTracker streams the data to. This is shown in the ITU-GazeTracker settings as port 66664. I calibrated ITU-GazeTracker, then tuned ComCap to the UDP output port, and started ITU-GazeTracker recording, but no data seemed to be transmitted. After checking settings, searching for information about ComCap and UDP capture, and retrying several times, and turned to searching for more information. Not knowing much about UDP, I searched the discussion forums and contacted the University of Missouri IAT services. A person there responded, but knew little more about UDP communications than I did. Finally, in one of the ITU-GazeTracker discussion forums, I spotted a post from a user stating that the UDP port setting in ITU-GazeTracker is ignored, and there is a default UDP port that is used irrespective of the setting. Using this port in ComCap, I started collecting UDP data.

The next day or so I received a reply from Xavier stating that ITU-GazeTracker definitely produced of log file of eyetracking data, and suggested I recheck the settings when I record, and look in the application directory where it was supposed to be saved. I replied with what Martin, Xavier's partner, had written about the log file not being implemented, and that UDP streaming was the only way to get data output. Xavier's reply stated that Martin was mistaken, and the feature was definitely implemented. I was starting to question who knew what, and asked Xavier for the ITU-GazeTracker source code. He supplied me with the link, and suggested I download the source code and recompile it. Doing so would require installing Microsoft Visual Studio for C#, which would require more time. Having been exposed to s seeming rift between the ITU-GazeTracker developers, and not wishing to go down yet another dead end, I first exhaustively searched all of the information in the discussion forums for any clue as to where the log file might be saved. All of the related posts by users pointed to the log feature being unimplemented.

Finally I downloaded and installed Microsoft Visual Studio for C#, loaded the project files for ITU-GazeTracker and rebuilt the project. I ran the new executable I had produced, calibrated, and started recording, and the log file was created. I contacted Xavier with this good news, and inquiring why his organization did not make the current build of ITU-GazeTracker available as a downloadable exe file. His response to me was an apology, but did not offer a good explanation to my question.

Data Analysis

Now that I finally had some data, I turned my attention to analyzing it with OGAMA, but this proved to be as difficult as working with ITU-GazeTracker, if not more. There is absolutely no manual or guide for OGAMA. There is online help, but it references OGAMA's API only – nothing about how to use its features.

After running OGAMA and looking through the features, I found an import wizard that allows importing data from a text file.

The import process was straight forward, similar to Excels text file import, guiding each step in the process but there are several fields OGAMA needs to have filled that are not present in the ITU-GazeTracker log file. The lack of documentation caused me to fumble through the process several times before finding the right options that would allow me to import without generating an error.

OGAMA writes the imported data to an SQL database and provides a raw data table view of the imported data. When I opened the raw data table, I noted that all of the time codes for each record in the table were zero. I guessed that OGAMA needed the time code in some other format, or perhaps relative to the starting time of the dataset. (ITU-GazeTracker saves a system time code in milliseconds – a rather large number). I tried recalculating the time code field in the log file by importing into Excel first, and setting the first records time code to 0, and basing the remaining record's time codes relative to that start time. This strategy did not help – OGAMA still zeroed out all the imported time codes.

After several variations of this strategy failed, I sent an email to the developer, Adrian, in Germany. Adrian responded a few days later, asking what version I was using, and suggesting I try again and send him the activity log OGAMA produces. (He also suggested I write a "connector" to ITU-GazeTracker using the OGAMA API, so that I could capture data directly from ITU-GazeTracker.) Following his suggestion I imported again, but the activity log was empty. I reported my results to Adrian attached the file ITU-GazeTracker log file I was attempting to import. Adrian replied the next day that he had discovered a bug in the OGAMA source code that prevented it from importing the ITU-GazeTracker log file time code field. He had fixed it, and uploaded a new version of OGAMA to try. As my time on this project had already run out, I have not had a chance to verify his claim, but am satisfied that such difficulties with OGAMA can be overcome.

Finally, a couple of issue that became evident along the way:

- ITU-GazeTracker occasionally locks up at the end of the calibration routine. I have yet to contact the developer about this issue because I have not been able to replicate the problem with any reliability.
- ITU-GazeTracker only logged one set of coordinates, whereas the developer had stated there would be two sets: raw eye position coordinates and calculated fixation coordinates. I did not have time to follow-up with the developer to determine what data was being logged.

Lessons Learned and Recommendations

Open source vs. proprietary systems

Commercially available proprietary systems offer turnkey integration with data collection and analysis processes. Training and maintenance costs are potentially lower than with open source systems. Initial startup costs are much higher with proprietary systems but these might be returned in ease of use and maintenance issues. The open source system was composed of separate analysis and tracking systems, which require some work to coordinate. In a busy lab, the resources required to run an open source eyetracking might make a higher cost and more integrated system economical.

Resources and costs

Commercially available proprietary systems are potentially much more expensive. For approximately $60 for a camera and infrared light source, and 70 hours of work, I believe I have achieved proof of concept using open source systems. But this proof of concept is not a usable system due to these factors:

- Camera used, the MS LifeCam VX1000 produced unstable images
- ITU-GazeTracker has issues that need to be worked out with the developer – the logged data format needs to be reconfirmed; occasionally locks up at the end of the calibration routine.
- OGAMA processes eyetracking data on static images. A system would need to be worked out to process dynamic moving images.
- No screen capture utility was tested
- ITU-GazeTracker was not tested running concurrently with a screen capture program and iSocial
- Testing an eyetracking system with people who have ASD

Open source support

Using open source requires working with developers to determine the state of the software – I would have contacted the developers much earlier had I known that features were not implemented, or buggy. Smaller communities of users mean less useful information is available.

The developers were eager to help when they learned of iSocial, though ITU-GazeTracker's author, Xavier, was busy writing his dissertation. Time lags between Europe and Missouri meant that sometimes replies were delayed.

Pulling together a usable eyetracking and analysis system

In my estimation, to develop this to an eyetracking system that approaches turnkey integration would require a high-resolution camera and a minimum of 40 hours of work. Work is necessary to integrate the new camera; and work with the developers of ITU-GazeTracker and OGAMA to settle unresolved problems; develop an experiment procedure for testing target participants.

I believe cost of running the system once it was developed would probably mean 10 minutes at the start of every trail, and another 20 minutes to process the data – maybe less once the processes is refined.

Directions for future investigation

1. Resolve camera image stability issues. Probably requires using a higher quality camera.
2. Resolve ITU–GazeTracker bugs and issues with developer.
3. Determine method for analyzing eyetracking data used with moving images. Flash might be one avenue: Flash can import eyetracking data and be used to create visual overlays with the recorded iSocial screen capture video
4. Create a process to use ITU–GazeTracker with iSocial's dynamic visual environment. This involves paring up a screen capture program with iSocial and ITU-GazeTracker running concurrently.
5. Test the system with real participants. Working with ASD students might introduce unknown issues: we don't know how they might react to the calibration process; we don't know if the presence of the camera may affect their behavior.

Summary

This open source eyetracking investigation contained significant challenges that were overcome enough to satisfy proof of concept, but work is still needed to create and usable better integrated system and research process. While open source eyetracking systems have smaller startup costs, the lack of integrated features may require more resources to run, maintain and train users. Open source systems test were in varying states of completion, and problems were encountered frequently. Small user bases make resolving problems difficult. On the other hand, developers seem eager to help with high profile research and establishing relationships with them is critical.