An Open Source Eye-gaze Interface: Expanding the Adoption of Eye-gaze in Everyday Applications

Craig Hennessey*
Mirametrix Research

Andrew T. Duchowski†
School of Computing, Clemson University

Abstract

There is no standard software interface in the eye-tracking industry, making it difficult for developers to integrate eye-gaze into their applications. The combination of high cost eye-trackers and lack of applications has resulted in a slow adoption of the technology. To expand the adoption of eye-gaze in everyday applications, we present an eye-gaze specific application programming interface that is platform and language neutral, based on open standards, easily used and extended and free of cost.

1 Introduction & Background

[1]

The use of eye-gaze as an interface is still rarely observed outside of research laboratories [Duchowski 2003]. While recent low-cost [Babcock and Pelz 2004; Li et al. 2006] and open source [San Agustin et al. 2009] initiatives have been proposed, high equipment cost and a lack of applications are holding back widespread adoption of eye-gaze in everyday applications. With no clear standard established, developers wanting to integrate eye-gaze into their applications must choose between two interface mechanisms, simple cursor following or a vendor specific application programming interface (API).

1.1 Cursor tracking

The simplest way to add eye-gaze to an application is to link the mouse cursor position to the eye-tracker point-of-gaze (POG), and then track the cursor's position. A number of applications, such as Dasher [Ward and MacKay 2002] and Stargazer [Hansen et al. 2008], currently use this technique. Its advantages are simplicity, availability, and no additional costs.

However, this approach lacks the additional functionality related to the POG such as a POG validity flag, binocular data (left and right eye POG), eye position information, fixation/saccade tracking and more. Furthermore, once the mouse cursor is tied to the POG, the mouse is no longer available for traditional input.

1.2 Vendor API

Until now, to achieve full eye-gaze application functionality, a vendor specific API was required. A full API usually provides complete eye-gaze information, including the left and right eye POG, head position, and data validity flags. In addition, vendors may provide the ability to customize the internal algorithms, such as the filtering and fixation detection methods.

One disadvantage of using a vendor supplied API is that proprietary libraries (`lib` or `dll`) must be included, which are typically compiled for a specific operating system and a specific programming language. These libraries are included as part of a software development kit (SDK) which must be purchased from the vendor. As every vendor has a different SDK, a software developer must acquire an SDK for each eye-tracker they wish to support.

Consequently, it is possible to develop a translator application that acts as an intermediary between the vendor's API and an application, as exemplified by the development of the Eye-Tracking Universal Driver (ETU-Driver),[2] one of COGAIN's Work Package 2 (WP2) deliverables [Bates et al. 2005; Bates and Špakov 2006]. Although this effort also suggested an XML-based universal API, the resultant ETU-Driver was wrapped into a COM object, limiting its portability across operating systems.

1.3 Open Eye-gaze API

In this paper we propose an open eye-gaze API that combines the best of the techniques described above. A summary of the proposed eye-gaze interface is shown in comparison with the traditional methods in Table 1.

The open eye-gaze API proposed here is based on the web-services API model. This model uses TCP/IP as the communication mechanism and the extensible mark-up language (XML) as the data format. Both TCP/IP and XML are open standards that are readily available on most operating systems and programming languages. Data and commands are encoded in plain text XML and transmit as simple strings over the TCP/IP communication layer, providing maximum transparency. The open eye-gaze API will be described in detail in the following section.

Table 1: *Comparison of eye-gaze interface techniques.*

Feature	Cursor	Vendor API	Open eye-gaze API
Platform neutral	Yes	No	Yes
Language neutral	Yes	No	Yes
Open standards	Yes	No	Yes
Proprietary data format	N/A	Yes	No
Cost	None	High*	None
Baseline feature set	X/Y	Full	Full
Extendable	No	No*	Yes

* Vendor dependent

2 Implementation

The open eye-gaze API uses a client-server architecture. The client application configures selected variables on the server such as the fixation POG or left and right eye positions, and initiates the data transmission sequence. The client then listens for data records sent

*craig@mirametrix.com
†duchowski@clemson.edu

[2] `http://www.cs.uta.fi/~oleg/etud.html`

Table 2: *Configuration variables for the open eye-gaze API.*

Data ID	Read/Write	Parameters	Type	Description
ENABLE_SEND_DATA	R/W	STATE	boolean	Start or stop data streaming
ENABLE_SEND_COUNTER	R/W	STATE	boolean	Enable record ID counter data
ENABLE_SEND_TIME	R/W	STATE	boolean	Enable time stamp data
ENABLE_SEND_TIME_TICK	R/W	STATE	boolean	Enable high resolution timer tick
TIME_TICK_FREQUENCY	R	FREQ	long long	Tick frequency (signed 64-bit integer)
ENABLE_SEND_POG_LEFT	R/W	STATE	boolean	Enable left eye point-of-gaze data
ENABLE_SEND_POG_RIGHT	R/W	STATE	boolean	Enable right eye point-of-gaze data
ENABLE_SEND_POG_FIX	R/W	STATE	boolean	Enable fixation point-of-gaze data
ENABLE_SEND_PUPIL_LEFT	R/W	STATE	boolean	Enable left eye image data
ENABLE_SEND_PUPIL_RIGHT	R/W	STATE	boolean	Enable right eye image data
SCREEN_SIZE	R	WIDTH	int	Screen width (pixels)
SCREEN_SIZE	R	HEIGHT	int	Screen height (pixels)
CAMERA_SIZE	R	WIDTH	int	Camera image width (pixels)
CAMERA_SIZE	R	HEIGHT	int	Camera image height (pixels)
PRODUCT_ID	R	VALUE	string	Product identifier
SERIAL_ID	R	VALUE	string	Device serial number
COMPANY_ID	R	VALUE	string	Manufacturer identifier
API_ID	R	MFG_ID	string	API vendor identity
API_ID	R	VER_ID	string	API version number
API_SELECT	R	MFG_ID?	string	List of vendors supported
API_SELECT	R	VER_ID?	string	List of versions supported
API_SELECT	W	STATE	int	Selected API

by the server each time a new POG is computed. Commands and records are formatted in XML strings which are transmitted over a TCP/IP connection, allowing communication between multiple computers. Using TCP rather than UDP ensures records are not lost and are received in the correct order.

2.1 XML Format

Data and commands are formatted using XML string fragments called elements. Each element is defined by an empty-element TAG that specifies the element type. An empty-element TAG is of the form <GET ... />, which is shorter than the start/end element format <GET>...</GET>. Only six XML tags are required for the open eye-gaze API and are listed in Table 3.

Table 3: *XML TAG identifiers in the open eye-gaze API*

Client	
TAG	Description
GET	Get a data variable or command
SET	Set a data variable or command
Server	
TAG	Description
ACK	Acknowledge a successful command
NACK	Acknowledge a failed command
CAL	Calibration result record
REC	Data result record

Any additional parameters in a data record or command are defined by attributes taking the form of name/value pairs, e.g.:

```
<GET ID="ENABLE_SEND_TIME" />
```

where the attribute is ID and the value is ENABLE_SEND_TIME.

As XML elements are read by the client and server it is possible that a partial element will be read if the reader is faster than the sender, or multiple elements may be read if the reader is slower. A partial and multiple record are shown below:

```
<GET ID="ENABLE_SE
<GET ID="COMPANY_ID" /><GET ID="API_ID" />
```

There is no identifier available to signal *end of transmission* for an open TCP connection and therefore to delimit records, a carriage return (CR), line feed (LF) pair (\r\n) is introduced, indicating the end of the string. The CRLF sequence is safe to use as the record delimiter as the XML specification disallows CRLF from appearing within the XML string.

2.2 Client / Server

The eye-tracker operates as the server and the application operates as the client. A TCP stream is opened between the client application and the eye-gaze server using the IP address of the server and an assigned port number (4242 is the default port value). Applications may be run on completely remote computers and connect to the eye-tracking server by using the appropriate remote server IP address. Multiple eye-trackers may also be run on the same machine by assigning different port numbers to the different servers (for example 4242 for tracker 1 and 4243 for tracker 2). A client application may then connect to multiple eye-trackers by simply opening two ports for communication (4242 and 4243 in this example).

The server can operate in three different modes: configuration, calibration and data transmission.

Configuration. In configuration mode the server responds to each XML element query (GET or SET TAG) with the appropriate XML (ACK or NACK TAG). This mode is used to select the variables in the requested data stream, query other eye-tracker variables and to initiate or end calibration and data transmission. An example of a configuration event where the COUNTER variable is enabled in the data stream would appear as follows:

```
CLIENT SEND: <SET ID="ENABLE_SEND_COUNTER" STATE="1" />
SERVER SEND: <ACK ID="ENABLE_SEND_COUNTER" STATE="1" />
```

The full list of configuration variables in the eye-gaze API are listed in Table 2. Camera and screen sizes are returned in pixel units. The POG X and Y eye position coordinates are normalized to the screen and camera image sizes respectively. For monocular

eye-trackers, the LEFT eye data structure should be used and the RIGHT eye data structure left cleared (zeroed).

Calibration. To reduce the complexity for application developers, the calibration of the eye-tracker is performed by the server but can be initiated by the client using the commands listed in Table 4. The calibration process includes the display of calibration markers, the actual calibration procedure, and the display of the results.

During calibration the server transmits calibration XML elements (CAL TAG and ID attribute value CALIB_RESULT_PT) after each calibration point is completed as shown in Table 5. The PT attribute indicates which calibration point was completed. After the entire calibration procedure is complete, the results of the calibration are returned with the CALIB_RESULT attribute, also shown in Table 5. The CALX?/CALY?, LX?/LY? and RX?/RY? results are returned in percentages of the screen (0-100%) and can be used to provide feedback on how well the calibration performed for the eye at the indicated point. The LV? and RV? flags provide an indicator of whether the calibration was successful at all.

Table 4: *Commands for calibration display and control.*

Data ID	Read/ Write	Param	Type	Description
CALIBRATE_START	R/W	STATE	Bool	Start or stop the calibration procedure
CALIBRATE_SHOW	R/W	STATE	Bool	Show or hide the default calibration window

Table 5: *Calibration data and results.*

Data ID	Param	Type	Description
CALIB_RESULT_PT	PT	int	Calibration point completed
CALIB_RESULT	CALX? CALY?	float	Calibration X/Y coordinate for point ?
CALIB_RESULT	LX? LY?	float	Left eye POG X/Y for point ?
CALIB_RESULT	LV?	int	Left eye valid flag for point ?
CALIB_RESULT	RX? RY?	float	Right eye POG X/Y for point ?
CALIB_RESULT	RV?	int	Right eye valid flag for point ?

? = the calibration point number

To illustrate the operation of the open eye-gaze API calibration procedure the following listing shows an entire calibration sequence on a 4 point calibration grid.

```
CLIENT SEND: <SET ID="CALIBRATE_SHOW" STATE="1" />
SERVER SEND: <ACK ID="CALIBRATE_SHOW" STATE="1" />
CLIENT SEND: <SET ID="CALIBRATE_START" STATE="1" />
SERVER SEND: <ACK ID="CALIBRATE_START" STATE="1" />
SERVER SEND: <CAL ID="CALIB_RESULT_PT" PT="1" />
SERVER SEND: <CAL ID="CALIB_RESULT_PT" PT="2" />
SERVER SEND: <CAL ID="CALIB_RESULT_PT" PT="3" />
SERVER SEND: <CAL ID="CALIB_RESULT_PT" PT="4" />
SERVER SEND: <CAL ID="CALIB_RESULT"
CALX1="0.10000" CALY1="0.08000"
LX1="0.00000" LY1="0.00000" LV1="0"
RX1="0.09881" RY1="0.09238" RV1="1"
CALX2="0.90000" CALY2="0.08000"
LX2="0.90595" LY2="0.08952" LV2="1"
RX2="0.88869" RY2="0.09905" RV2="1"
CALX3="0.10000" CALY3="0.92000"
LX3="0.08631" LY3="0.89143" LV3="1"
RX3="0.09881" RY3="0.92952" RV3="1"
CALX4="0.90000" CALY4="0.92000"
LX4="0.89524" LY4="0.90095" LV4="1"
RX4="0.89583" RY4="0.92190" RV4="1" />
```

Data Transmission. When in free running data transmission mode, the server transmits an XML record (REC TAG) after each new POG computation. The XML record contains attributes that match the configured data requested in the configuration mode. Attributes selected with commands in Table 2 are listed in Table 6.

Synchronizing recorded eye-gaze with external data such as software events and biological data such as EMG and EEG can be performed using the TIME_TICK variable, which is a measure of elapsed CPU timer ticks and is equal to the output of a high resolution timing function. Conversion from ticks to seconds is achieved by dividing by the TIME_TICK_FREQUENCY value from Table 2.

Unfiltered POG estimates are available by reading the left and right eye POG data (LPOG? and RPOG? respectively). This data can then be processed using any desired algorithm for filtering and extracting fixations. To avoid the need for the application developer to implement their own fixation detector, a fixation data record is also available (FPOG?). The fixation data record is generated by whichever fixation algorithm (e.g., position-variance, velocity-based, etc.) is implemented by the eye-tracker, and provides the X/Y fixation position on the screen, the start time and duration in seconds, a valid flag, and an identifier indicating the fixation's ID.

Table 6: *Data attributes available in the data record.*

Param	Type	Description
CNT	int	Sequence counter for data packets
TIME	float	Elapsed time in seconds since last system initialization or calibration
TIME_TICK	long long	Tick count (signed 64-bit integer)
LPOGX	float	Left point-of-gaze X
LPOGY	float	Left point-of-gaze Y
LPOGV	int	Left point-of-gaze valid flag
RPOGX	float	Right point-of-gaze X
RPOGY	float	Right point-of-gaze Y
RPOGV	int	Right point-of-gaze valid flag
FPOGX	float	Fixation point-of-gaze X
FPOGY	float	Fixation point-of-gaze Y
FPOGS	float	Fixation start (seconds)
FPOGD	float	Fixation duration (time since fixation start (seconds))
FPOGID	int	Fixation number ID
FPOGV	int	Fixation point-of-gaze valid flag
LPCX	float	Left eye pupil center X
LPCY	float	Left eye pupil center Y
LPD	float	Left eye pupil diameter
LPS	float	Left eye pupil distance (unit less, from calibration position)
LPV	int	Left eye pupil image valid
RPCX	float	Right eye pupil center X
RPCY	float	Right eye pupil center Y
RPD	float	Right eye pupil diameter
RPS	float	Right eye pupil distance (unit less, from calibration position)
RPV	int	Right eye pupil image valid

To illustrate the operation of the open eye-gaze API, the following listing shows the configuration of the data record and begins data transmission. The data record includes the record counter and the fixation POG. For many applications this is the only data required for basic eye-gaze operation.

```
CLIENT SEND: <SET ID="ENABLE_SEND_COUNTER" STATE="1" />
SERVER SEND: <ACK ID="ENABLE_SEND_COUNTER" STATE="1" />
```

```
CLIENT SEND: <SET ID="ENABLE_SEND_POG_FIX" STATE="1" />
SERVER SEND: <ACK ID="ENABLE_SEND_POG_FIX" STATE="1" />
CLIENT SEND: <SET ID="ENABLE_SEND_DATA" STATE="1" />
SERVER SEND: <ACK ID="ENABLE_SEND_DATA" STATE="1" />
SERVER SEND: <REC CNT="72" FPOGX="0.5065" FPOGY="0.4390"
FPOGD="0.078" FPOGID="468" FPOGV="1"/>
SERVER SEND: <REC CNT="73" FPOGX="0.5071" FPOGY="0.4409"
FPOGD="0.094" FPOGID="468" FPOGV="1"/>
SERVER SEND: <REC CNT="74" FPOGX="0.5077" FPOGY="0.4428"
FPOGD="0.109" FPOGID="468" FPOGV="1"/>
```

2.3 API Extensions

The open eye-gaze API outlined in this paper corresponds to version 1.0 of the generic eye-gaze interface, identified by the `MFG_ID="generic"` and `VER_ID="1.0"` variables. Further collaborative efforts to improve the open eye-gaze API interface will correspond to increasing API version numbers. The API may be extended to include head mounted, or 3D eye-tracking specific variables [Hennessey and Lawrence 2008] by simply adding new attributes to the XML command and data tables.

Eye-tracking vendors should always at least support the generic, 1.0, interface, but may also add their own unique additions to the API under their own `MFG_ID` identifier.

The supported APIs for a given eye-tracker can be listed and selected using the `API_SELECT` command from Table 2. An example listing of available APIs for an eye-tracker is shown below.

```
CLIENT SEND: <GET ID="API_SELECT" />
SERVER SEND: <ACK ID="API_SELECT" MFG_ID0="generic"
VER_ID0="1.0" MFG_ID1="Mirametrix" VER_ID1="1.0" />
CLIENT SEND: <SET ID="API_SELECT" STATE="1" />
SERVER SEND: <ACK ID="API_SELECT" STATE="1" />
```

Future versions of the open eye-gaze API, as well as example source code can be found at the following link: `http://www.mirametrix.com/eye-gaze-api.html`.

3 Applications

The eye-gaze interface was tested with a binocular portable eye-gaze tracker for evaluation. The interface was used for two different applications at two institutions, one integrating with EEG recording, and another using projection screen eye-tracking. The projection screen system's (see Figure 1) application was developed entirely within the Qt toolkit under Windows XP in MS Visual Studio 2008. Qt provides an API to underlying TCP/IP (e.g., `QtcpSocket`) as well XML parsing functionality (e.g., `QtXmlSimpleReader`), and is easily portable to Linux and Mac OS X. The application was developed within one 16 week semester.

4 Conclusions

The presented open standard eye-gaze API provides the basic functionality required to integrate eye-gaze into applications. By basing the API on established and well known standards such as TCP/IP and XML, the API is accessible on most operating system platforms and programming languages without any additional libraries or cost. The API provides a standard format for eye-gaze data and minimizes the need for application developers to deal with eye-gaze specific issues such as calibration and fixation detection. The API also provides a mechanism for accommodating future improvements and enhancements to the interface, allowing eye-trackers and applications to support multiple API versions. The open eye-gaze API can be used by commercial, low cost and open-source eye-tracker developers to quickly support all eye-gaze aware applications based on the API.

The proposed standard provides developers with a simple and effective method for adding eye-gaze to their applications. Should eye-tracking companies adopt this standard, the ensuing number of applications could advance eye-gaze as the next big thing in human computer interaction.

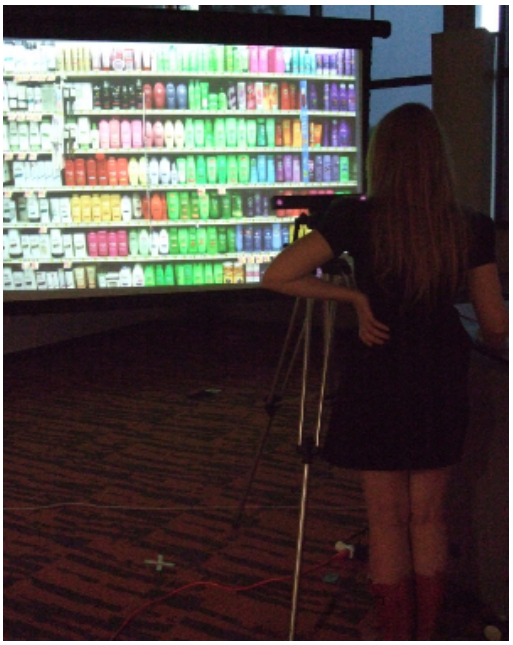

Figure 1: *Consumer experience lab.*

References

BABCOCK, J. S. AND PELZ, J. B. 2004. Building a lightweight eyetracking headgear. In *ETRA '04: Proceedings of the 2004 symposium on Eye tracking research & applications*. ACM, New York, NY, USA, 109–114.

BATES, R., ISTANCE, H., AND ŠPAKOV, O. 2005. D2.2 Requirements for the Common Format of Eye Movement Data. Tech. Rep. IST-2003-511598: Deliverable 2.2, Communication by Gaze Interaction (COGAIN).

BATES, R. AND ŠPAKOV, O. 2006. D2.3 Implementation of COGAIN Gaze Tracking Standards. Tech. Rep. IST-2003-511598: Deliverable 2.3, Communication by Gaze Interaction (COGAIN).

DUCHOWSKI, A. T. 2003. *Eye Tracking Methodology: Theory and Practice*. Springer-Verlag.

HANSEN, D. W., SKOVSGAARD, H. H. T., HANSEN, J. P., AND MØLLENBACH, E. 2008. Noise tolerant selection by gaze-controlled pan and zoom in 3d. In *ETRA '08: Proceedings of the 2008 symposium on Eye tracking research & applications*. ACM, New York, NY, USA, 205–212.

HENNESSEY, C. AND LAWRENCE, P. 2008. 3d point-of-gaze estimation on a volumetric display. In *ETRA '08: Proceedings of the 2008 symposium on Eye tracking research & applications*. ACM, New York, NY, USA, 59–59.

LI, D., BABCOCK, J., AND PARKHURST, D. J. 2006. openeyes: a low-cost head-mounted eye-tracking solution. In *ETRA '06: Proceedings of the 2006 symposium on Eye tracking research & applications*. ACM, New York, NY, USA, 95–100.

SAN AGUSTIN, J., SKOVSGAARD, H., HANSEN, J. P., AND HANSEN, D. W. 2009. Low-cost gaze interaction: ready to deliver the promises. In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*. ACM, New York, NY, USA, 4453–4458.

WARD, D. J. AND MACKAY, D. J. C. 2002. Fast hands-free writing by gaze direction. *Nature 418,* 6900, 838.