Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-based Interaction

Moritz Kassner
Pupil Labs UG
Berlin, Germany
moritz@pupil-labs.com

William Patera
Pupil Labs UG
Berlin, Germany
will@pupil-labs.com

Andreas Bulling
Max Planck Institute for Informatics
Perceptual User Interfaces Group
Saarbrücken, Germany
andreas.bulling@acm.org

 ACM 978-1-4503-3047-3/14/09$ 15.00.
http://dx.doi.org/10.1145/2638728.2641695

Abstract
In this paper we present Pupil – an accessible, affordable, and extensible open source platform for pervasive eye tracking and gaze-based interaction. Pupil comprises 1) a light-weight eye tracking headset, 2) an open source software framework for mobile eye tracking, as well as 3) a graphical user interface to playback and visualize video and gaze data. Pupil features high-resolution scene and eye cameras for monocular and binocular gaze estimation. The software and GUI are platform-independent and include state-of-the-art algorithms for real-time pupil detection and tracking, calibration, and accurate gaze estimation. Results of a performance evaluation show that Pupil can provide an average gaze estimation accuracy of 0.6 degree of visual angle (0.08 degree precision) with a processing pipeline latency of only 0.045 seconds.

Author Keywords
Eye Movement; Mobile Eye Tracking; Wearable Computing; Gaze-based Interaction

ACM Classification Keywords
C.3 [Special-purpose and application-based systems]: Real-time and embedded systems.; H.5 [Information Interfaces and Presentation]: User Interfaces.

Introduction

Recent advances in head-mounted eye tracking and automated eye movement analysis point the way toward unobtrusive eye-based human-computer interfaces that are pervasively usable in everyday life. This new paradigm is called pervasive eye tracking - continuous eye monitoring and analysis [2]. The ability to track and analyze eye movements anywhere and at any time will enable new research to develop and understand visual behavior and eye-based interaction in daily life settings.

Commercially available head-mounted eye tracking systems are robust and provide useful features to customers in industry and research, such as for marketing studies, website analytics, or research studies. However, commercial systems are expensive, therefore typically used by specialized user groups, and rely on closed source hardware and software. This limits the potential scale and application areas of eye tracking to expert users and inhibits user-driven development, customization, and extension. Open source software (OSS) eye trackers have emerged as low cost alternatives to commercial eye tracking systems using consumer digital camera sensors and open source computer vision software libraries [1, 8, 13, 9, 10, 11, 4, 16]. The OSS route enables users to rapidly develop and modify hardware and software based on experimental findings [2].

We argue that affordability does not necessarily align with accessibility. In this paper we define accessible eye tracking platforms to have the following qualities: open source components, modular hardware and software design, comprehensive documentation, user support, affordable price, and flexibility for future changes. Towards this goal we describe Pupil, a mobile eye tracking headset and an open source software platform as an accessible, affordable, and extensible tool for pervasive eye tracking research. We explain the design motivation, provide an in depth technical description of both hardware and software, and provide an analysis of accuracy and performance of the system.

Figure 1: Front rendering of the Pupil Pro headset (rev 20) showing the frame, tiltable scene camera and adjustable eye camera.

System Overview

Pupil is a mobile eye tracking headset with one scene camera and one infrared (IR) spectrum eye camera for dark pupil detection. Both cameras connect to a laptop, desktop, or mobile computer platform via high speed USB 2.0. The camera video streams are read using Pupil Capture software for real-time pupil detection, gaze mapping, recording, and other functions.

Figure 2: Rendering demonstrating the range of motion of the world camera (scene camera) mount and connection to the frame.

Figure 3: Rendering demonstrating the range of motion of the eye camera mount and connection to the frame.

System Design Objectives

We made several design decisions while satisfying a number of factors to balance ergonomic constraints with performance. Pupil leverages the rapid development cycle and scaling effects of consumer electronics - USB cameras and consumer computing hardware - instead of using custom cameras and computing solutions. Pupil headsets are fabricated using Selective Laser Sintering (SLS) instead of established fabrication methods like injection molding. This rapid fabrication process accommodates frequent design changes, comparable to the continuous development of Pupil software. We employed modular design principles in both hardware and software to enable modifications by users. Pupil software is open source and strives to build and support a community of eye tracking researchers and developers.

Pupil Headset Design and Hardware

Factors that influence the headset design are: mobility, modularity and customization, minimizing visual obstruction and weight, accommodation of various facial geometries, minimization of headset movement due to slippage and deformation, durability, and wear comfort.

Headset

The Pupil headset is made up of three modules: frame, scene camera mount, and eye camera mount.

Frame

The frame was designed based on a 3D scan of a human head and iteratively refined to accommodate physiological variations between users. We developed a novel design process where we apply specific forces to deform the headset model using Finite Element Analysis (FEA).

We use Finite Element Analysis (FEA) to apply forces that push the earpieces of the frame together. We print the resulting deformed geometry from the simulation. When the user puts on the headset, the earpieces of the frame are pushed apart, and the entire frame flexes open. This process ensures that cameras align as designed when the headset is worn and results in a comfortable, snug fitting, and lightweight (9g) frame.

Camera Mounts

Camera mount geometries are hosted in a Git repository. By releasing the mount geometry we automatically document the interface, allowing users to develop their own mounts for cameras of their choice. The scene camera mount connects to the frame with a snap fit toothed ratcheting system that allows for radial adjustment within the users vertical field of vision (FOV) along a transverse axis within a 90 degree range. The eye camera mount is an articulated adjustable arm accommodating variations in users eyes and face geometries. The camera mount attaches to the frame along a snap fit sliding joint. The eye camera orbits on a ball joint that can be fixed by tightening a single screw.

Cameras

Pupil uses USB cameras that comply with the UVC standard. Other UVC compliant cameras can be used with the system as desired by the user. The Pupil headset can be used with other software that supports the UVC interface. Pupil can be easily extended to use two eye cameras for binocular setups and more scene cameras.

Computing Device

Pupil works in conjunction with standard multipurpose computers: laptop, desktop, or tablet. Designing for user supplied recording and processing hardware introduces a source for compatibility issues and requires more setup effort for both users and developers. However, enabling the user to pair the headset with their own computing

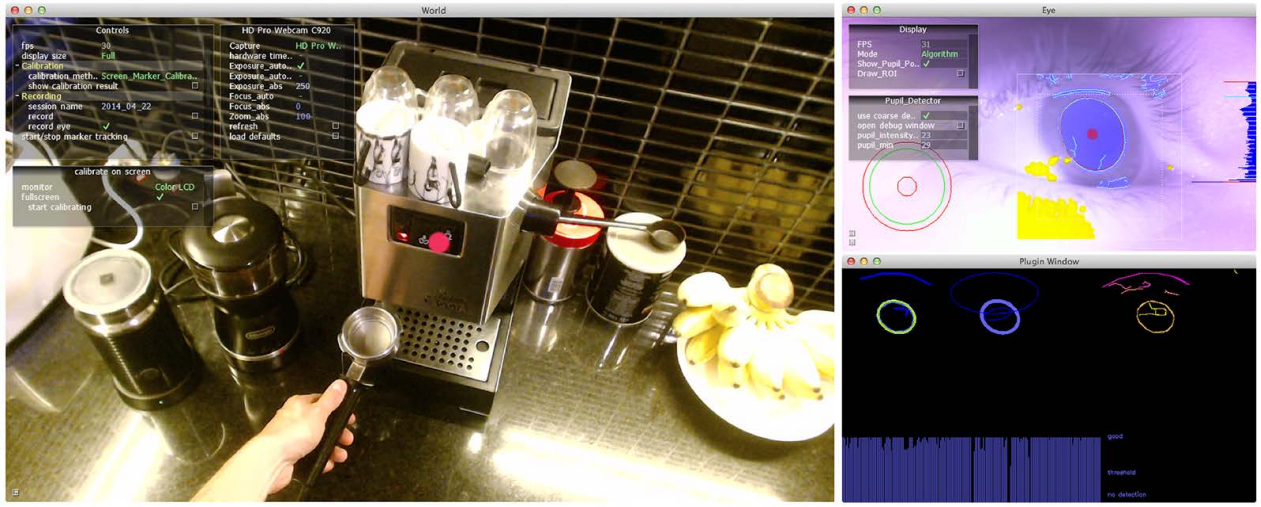

Figure 4: Screen capture of Pupil Capture. Left) World window displays real-time video feed of the user's FOV from the scene camera along with GUI controls for the world camera and plugins. Red circle is the gaze position of the user. Top Right) Eye window displays real-time video feed of the user's eye. Bottom Right) Plugin window plugins can also spawn their own windows. Shown here is a plugin to visualize the pupil detection algorithm.

platform makes Pupil a multipurpose eye tracking and analysis tool. Pupil is deployable for lightweight mobile use as well as more specialized applications like: streaming over networks, geotagging, multi-user synchronization; and computationally intensive applications like real time 3D reconstruction and localization.

Pupil Software

Pupil software is divided into two main parts, Pupil Capture and Pupil Player. Pupil Capture runs in real-time to capture and process images from the two (or more) camera video streams. Pupil Player is used to playback and visualize video and gaze data recorded with Pupil Capture. Source code is written in Python and C. Pupil software can be run from source on Linux, MacOS, and Windows or executed as a bundled double click application on Linux and MacOS.

Pupil Detection Algorithm

The pupil detection algorithm locates the dark pupil in the IR illuminated eye camera image (see Figure 6). In most environments, the algorithm is robust against reflections in the pupil area. Furthermore, the algorithm does not depend on the corneal reflection for detection, and works with users who wear contact lenses and eyeglasses.

The eye camera image is first converted to grayscale. The initial region estimation of the pupil is found via the

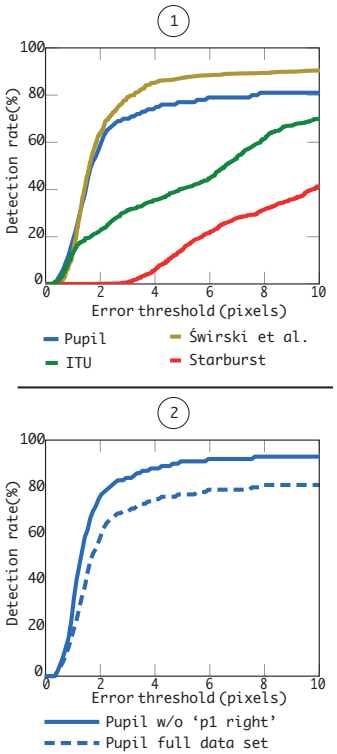

Figure 5: 1) Benchmark comparison 4 pupil detection algorithms. (Figure data for algorithms other than Pupil used with permission of Swirski.) 2) Pupil's detection algorithm comparison between full benchmark dataset and subset.

strongest response for a center-surround feature as proposed by Swirski et al. [15] within the image.

Fig 6-1) We use Canny [3] to find contours in the image and filter edges based on neighboring pixel intensity. **Fig 6-2**) We then look for darker areas (blue region) using a user-defined offset of the lowest spike in the histogram of pixel intensities in the eye image. **Fig 6-3**) We filter remaining edges to exclude those stemming from spectral reflections - yellow region. Remaining edges are extracted into into contours using connected components [14]. **Fig 6-4**) Contours are filtered and split into sub-contours based on criteria of curvature continuity. **Fig 6-5**) Candidate pupil ellipses are formed using ellipse fitting [5] onto a subset of the contours looking for good fits in a least square sense, major radii within a user defined range, and a few additional criteria. A combinatorial search looks for contours that can be added as support to the candidate ellipses. The results are evaluated based on the ellipse fit of the supporting edges and the ratio of supporting edge length and ellipse circumference (using Ramanujans second approximation [7]). We call this ratio "confidence". **Fig 6-6**) If the best results confidence is above a threshold the algorithm reports this candidate ellipse as the ellipse defining the contour of the pupil. Otherwise the algorithm reports that no pupil was found.

Figure 5 shows a performance comparison between Pupil's pupil detection algorithm, the stock algorithm proposed by Swirski et al., the ITU gaze tracker, and Starburst on the benchmark dataset by Swirski et al. [15]. As error measure we used the Hausdorff distance between the detected and hand-labeled pupil ellipses [15]. We additionally conducted a test excluding the dataset p1-right, that contains eye images recorded at the most extreme angles, as those do not occur using Pupil hardware.

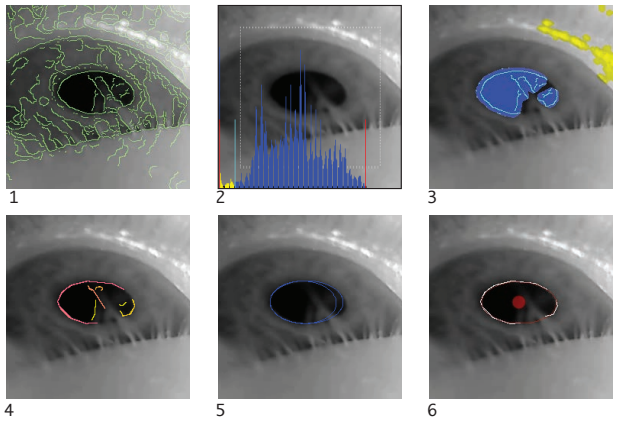

Figure 6: Visualization of pupil detection algorithm.

Fig 5-2 is a more accurate demonstration of how Pupil performs in real life conditions. Pupil achieves a detection rate of 80% with an error threshold of 2 pixels; at 5 pixels error detection rate increases to 90%.

Gaze Mapping
Mapping pupil positions from eye to scene space is implemented with a transfer function consisting of two bivariate polynomials of adjustable degree. The user specific polynomial parameters are obtained by running one of the calibration routines using screen markers, manual markers or natural features. Calibration and mapping functions are abstracted and the underlying models can easily be modified and replaced if needed.

Surface Detection
Pupil Capture can detect planar reference surfaces in the scene using markers. There are 64 unique markers. A minimum of 1 marker is required to define a surface. Gaze positions are mapped into the reference surface coordinate

system using homographic transformations between the scene camera plane and reference surface plane.

Recording
Pupil Capture can record the scene and eye videos, associated frame timestamps, detected pupil and gaze position data, and additional user activated plugin data.

Streaming
Pupil Capture can send real-time gaze and pupil information as well as plugin generated data via ZMQ to other applications and network enabled devices. Please see further documentation on our wiki: https://github.com/pupil-labs/pupil/wiki/Pupil-Capture#pupil-server

Plugin Structure
Even a standard feature, like recording a video, is abstracted as a plugin. This level of abstraction and modularity allows for users to develop their own tools even for low level functionality. In Pupil Capture plugins can be launched in either world or eye processes. The modular structure makes it easy to for users to test out different methods at runtime and for developers to extend software without breaking existing functionality. Plugins have the ability to create their own GUI within the process window, access to shared memory like gaze data, and spawn their own windows. Please see our wiki for further documentation on developing plugins: https://github.com/pupil-labs/pupil/wiki/Plugin-Guide

Performance Evaluation

Spatial Accuracy and Precision
As performance metrics for spatial accuracy and precision of the system we employ the metrics defined in COGAIN Eye tracker accuracy terms and definitions [12]

"**Accuracy** was calculated as the average angular offset (distance) (in degrees of visual angle) between fixations locations and the corresponding locations of the fixation targets."

"**Precision** was calculated as the Root Mean Square (RMS) of the angular distance (in degrees of visual angle) between successive samples to `xi, yi to xi+1,yi+1`."

We evaluated the performance in an indoor environment using the following procedure: A subject wearing the Pupil Pro eye tracker sat approximately 0.5m away from a 27 inch computer monitor. The eye tracker was calibrated using the standard 9-point screen marker based calibration routine in full screen mode. After calibration the subject was asked to fixate on a marker on the monitor. The marker sampled 10 random positions for 1.5 seconds and then revisited each of the 9 calibration positions. While the marker was displayed, gaze samples and the position of the marker detected in the scene camera were recorded. The data was then correlated into gaze point and marker point pairs based on minimal temporal distance. Gaze point marker point pairs with a spatial distance of more than 5 degrees angular distance were discarded as outliers as they do not correlate to system error but human error (no fixation, blink) [6].

Accuracy and precision were calculated in scene camera pixel space and converted into degrees of visual angle based on camera intrinsics.

This procedure was repeated with eight different subjects to reflect a more diverse pool of physiologies. The test was conducted with a Pupil Pro eye tracker revision 021. The test used Pupil Capture software, version 0.3.8 running on Mac OS. The test routine is part of Pupil Capture releases, starting with version 0.3.8 and available to all users.

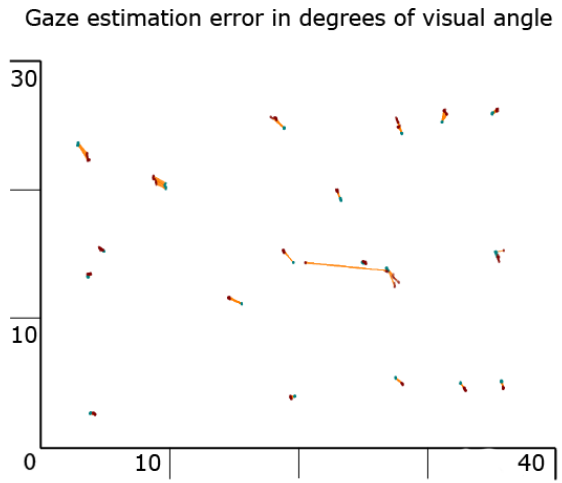

Figure 7: Results of accuracy test for one user. Marker target in green. Gaze point in red. Correspondence error in orange. Notice sample point with big error - during this sample period, the subject was not looking at the marker at the beginning of the test.

Results
Under ideal conditions 0.6 degrees of accuracy and 0.08 degrees of precision are achieved.

Temporal Accuracy, Latency and Precision

A second critical aspect are the temporal characteristics of the pupil eye tracking system.

Stream Synchronization
Timestamping is crucial for stream synchronization because data is obtained from two independent free running video sources. Additionally timestamps are used for correlation with additional external experiment or sensor data. Thus we strive to obtain a timestamp that is closest to the timepoint of data collection (in our case camera sensor exposure).

Pupil capture has two image timestamp implementations:

When the imaging camera is known to produce valid hardware timestamps, Pupil Capture uses these hardware image timestamps. These timestamps have high precision and accuracy as they are taken at the beginning of sensor exposure by the camera and transmitted along with the image data. The hardware timestamp accuracy exceeds our measurement capabilities. The variation of exposure times (jitter) reported by the hardware timestamps we measure by calculating the standard deviation of 1400 successively take frame times. It is 0.0004s for the world camera and 0.0001s for the eye camera.

Hardware timestamping is implemented in the Linux version of Pupil Capture and supported by both cameras of the Pupil Pro headset revision 020 and up. When Pupil Capture runs on an OS without timestamp video driver support or does not recognize the attached cameras as verified hardware timestamp sources, Pupil Capture uses software timestamps as a fallback. The timestamp is taken when the driver makes the image frame available to the software. Software timestamps are by nature of worse accuracy and precision, as they are taken after exposure, readout, image transfer, and decompression of the image on the host side. These steps take an indeterministic amount of time, which makes it impossible to accurately estimate time of exposure. Accuracy and precision depend on the camera and video capture driver.

System Latency
For real-time applications the full latency of Pupil is of great concern. We obtain processing times of functional groups in the Pupil hardware and software pipeline by

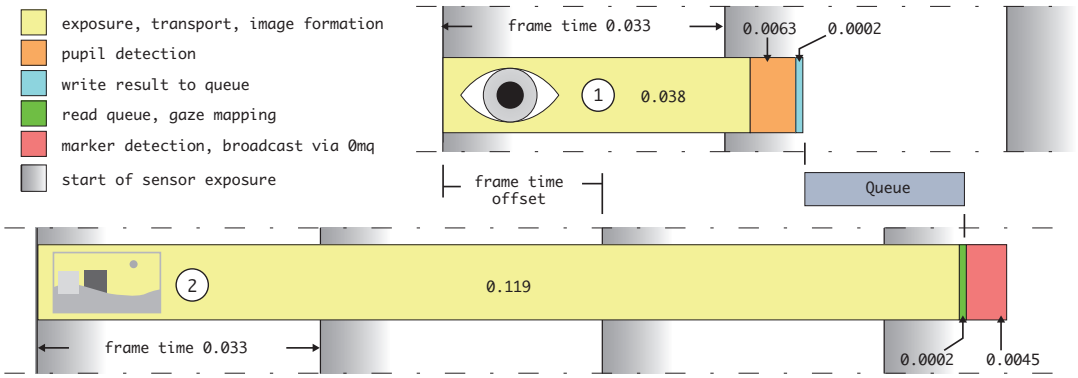

Figure 8: System latency diagram shows 1) eye process and 2) scene processes and times between sensor exposure and key points in the processing pipeline. Gray bars represent video streams of each process with vertical divisions as individual image frames within the streams. Color bars for key functions annotated with processing times.

calculating the time between sensor exposure and key points in the workflow. These temporal factors in signal delay are presented in Figure 8. All measurements were taken using Pupil Pro headset rev022 connected to a Lenovo X201 laptop with an Intel Core i7 620M CPU running Ubuntu 12.04. It should be noted that in real-time applications synchronicity of data is sacrificed for recency of data. The eye process pipeline is about 1/3 the latency of the world process pipeline. Therefore, we choose to broadcast the eye information as soon as it becomes available instead of waiting for the temporally closest scene image to become available.

Using the most recent data only makes sense for real-time applications. No sacrifices are made for any offline correlation of data employed by calibration, testing, or playback of recorded data. Furthermore, this approach does not prevent the user from obtaining accurate temporal correlation in real-time or offline applications. With this approach we can characterize the system latency for the eye pipeline, world pipeline separately:

- Total latency of the eye processing pipeline from start of sensor exposure to availability of pupil position: 0.045s (measured across 1400 samples with a standard deviation of 0.003 s)
- Total latency of the world pipeline including the eye measurement from start of sensor exposure to broadcast of pupil, gaze, and reference surface data via network: 0.124 s (measured across 1200 samples with a standard deviation of 0.005 s)

Minimum Hardware Requirements

The Pupil eye tracking system works with a traditional multipurpose computer - laptop, desktop, or tablet. It is therefore important to determine the minimum hardware specifications required to run Pupil Capture software in real-time. We tested the performance using a 11 inch Macbook Air (2010 model) with 2 GB of RAM and an Intel Core2Duo SU9400 dual core CPU. The software version used was Pupil Capture v0.3.8. The OS used on the machine specified above was Ubuntu 13.10.

Our performance test demonstrates that the system's CPU load never went above 90 percent, using the above hardware running Pupil Capture in recording mode, pupil detection at 30 fps and the world camera capture at 24 fps. The hardware setup was selected because it represents a portable computing platform with limited computing power. Any computer with a Intel "i" series processor or equivalent will have sufficient CPU resources, when comparing CPU benchmarks. Pupil Capture relies on several libraries to do video decompression/compression, image analysis, and display with platform specific implementations and efficiencies. Therefore, our test using a Linux distribution can not be generalized to Windows or MacOS. We found that requirements were similar for MacOS 10.8 and above. We did not establish Windows hardware requirements at the time of writing.

Conclusion

In order to further advance in eye tracking and to support the pervasive eye tracking paradigm, we will require accessible, affordable, and extensible eye tracking tools. We have developed Pupil as a contribution to the eye tracking research community. Pupil is already used in a wide range of disciplines and has developed a community of researchers and developers. Current limitations to the system are parallax error and tracking robustness in IR rich environments. Both Pupil software and hardware are under active development. Future developments will focus on hardware and software in parallel. The next big steps planned for Pupil are to improve mobility, implement real-time pose tracking and scene mapping, simplify user experience, and improve pupil tracking.

Links

1. Pupil mobile eye tracking headset: http://pupil-labs.com/pupil
2. Pupil open source code repository: http://github.com/pupil-labs/pupil
3. Pupil User Guides: https://github.com/pupil-labs/pupil/wiki/
4. Pupil user group forum: http://groups.google.com/forum/#!forum/pupil-discuss
5. Pupil open source headset mount repository https://code.google.com/p/pupil/source/browse/?repo=hardware
6. Pupil on arxiv.org http://arxiv.org/abs/1405.0006

References

[1] Babcock, J. S., and Pelz, J. B. Building a lightweight eyetracking headgear. In *Proceedings of the 2004 symposium on Eye tracking research & applications*, ACM (2004), 109–114.

[2] Bulling, A., and Gellersen, H. Toward Mobile Eye-Based Human-Computer Interaction. *IEEE Pervasive Computing 9*, 4 (2010), 8–12.

[3] Canny, J. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 6 (1986), 679–698.

[4] Ferhat, O., Vilariño, F., and Sánchez, F. J. A cheap portable eye-tracker solution for common setups.

Journal of Eye Movement Research 7, 3 (2014), 2:1–10.

[5] Fitzgibbon, A. W., Fisher, R. B., et al. A buyer's guide to conic fitting. *DAI Research paper* (1996).

[6] Global, T. Accuracy and precision test method for remote eye trackers, 2011.

[7] Hardy, G., Seshu Aiyar, P., and Wilson, B. Collected papers of srinivasa ramanujan. *Cambridge, London* (1927).

[8] Li, D., Babcock, J., and Parkhurst, D. J. openeyes: a low-cost head-mounted eye-tracking solution. In *Proceedings of the 2006 symposium on Eye tracking research & applications*, ACM (2006), 95–100.

[9] Lukander, K., Jagadeesan, S., Chi, H., and Müller, K. Omg!: A new robust, wearable and affordable open source mobile gaze tracker. In *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services*, ACM (2013), 408–411.

[10] Mantiuk, R., Kowalik, M., Nowosielski, A., and Bazyluk, B. Do-it-yourself eye tracker: Low-cost pupil-based eye tracker for computer graphics applications. In *Advances in Multimedia Modeling*. Springer, 2012, 115–125.

[11] Mardanbegi, D. Haytham gaze tracker: Diy hardware and open source software, March 2013.

[12] Mulvey, F. Woking copy of definitions and terminology for eye tracker accuracy and precision, 2010.

[13] Ryan, W. J., Duchowski, A. T., and Birchfield, S. T. Limbus/pupil switching for wearable eye tracking under variable lighting conditions. In *Proceedings of the 2008 symposium on Eye tracking research & applications*, ACM (2008), 61–64.

[14] Suzuki, S., et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing 30*, 1 (1985), 32–46.

[15] Świrski, L., Bulling, A., and Dodgson, N. Robust real-time pupil tracking in highly off-axis images. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ACM (2012), 173–176.

[16] Topal, C., Gerek, O. N., and Doğan, A. A head-mounted sensor-based eye tracking device: Eye touch system. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, ETRA '08, ACM (New York, NY, USA, 2008), 87–90.