EyeRec: An Open-source Data Acquisition Software for Head-mounted Eye-tracking

Thiago Santini, Wolfgang Fuhl, Thomas Kübler and Enkelejda Kasneci
Perception Engineering Group, University of Tübingen, Tübingen, Germany

Keywords: Eye Movements, Recording Software, Pupil Detection, Calibration, Open-source, Data Acquisition.

Abstract: Head-mounted eye tracking offers remarkable opportunities for human computer interaction in dynamic scenarios (e.g., driving assistance). Although a plethora of proprietary software for the acquisition of such eye-tracking data exists, all of them are plagued by a critical underlying issue: their source code is not available to the end user. Thus, a researcher is left with few options when facing a scenario in which the proprietary software does not perform as expected. In such a case, the researcher is either forced to change the experimental setup (which is undesirable) or invest a considerable amount of time and money in a different eye-tracking system (which may also underperform). In this paper, we introduce *EyeRec*, an open-source data acquisition software for head-mounted eye-tracking. Out of the box, *EyeRec* offers real-time state-of-the-art pupil detection and gaze estimation, which can be easily replaced by user implemented algorithms if desired. Moreover, this software supports multiple head-mounted eye-tracking hardware, records eye and scene videos, and stores pupil and gaze information, which are also available as a real-time stream. Thus, *EyeRec* can be an efficient means towards facilitating gazed-based human computer interaction research and applications. Available at: www.perception.uni-tuebingen.de

1 INTRODUCTION

In the past two decades, the number of researchers using eye trackers has grown enormously (Holmqvist et al., 2011). These researchers stem from several distinct fields (Duchowski, 2002). For instance, eye tracking has been employed from simple and fixed scenarios (e.g., language reading (Holsanova et al., 2006; Rayner, 1998)) to complex and dynamic cases (e.g., driving (Kasneci et al., 2014)). Naturally, these distinct fields usually have specific needs, which have lead to the spawning of several proprietary systems with different capabilities. In fact, Holmqvist et al. (Holmqvist et al., 2011) report that they were able to find 23 companies selling video-based eye-tracking systems in 2009. Typically, these commercial systems rely on closed-source software, offering their eye tracker bundled with their own software solutions. Although some companies offer technical transparency, satisfying researchers that want to know how data are gathered and analyzed by the proprietary software, one issue persists: what to do if the software fails to perform as desired. Facing such an issue, a researcher has few options: 1) changing the experimental scenario until the software performs properly, which is clearly undesirable and counterproductive, or 2) switching to a different eye-tracking system, which not only requires a considerable amount of time and money but also offers no guarantees that the alternative system will perform as desired. Few open-source alternatives do exist, such as openEyes (Li et al., 2006), PyGaze (Dalmaijer et al., 2014), and the very promising Pupil (Kassner et al., 2014); however, these focus on their own eye trackers or depend on existing APIs from manufacturers.

In this paper, we introduce *EyeRec*, an open-source data acquisition software for head-mounted eye tracking. In fact, we started with the development of *EyeRec* as we found the pupil detection algorithm of a proprietary software of a mobile head-mounted eye tracker to be underperforming in outdoor scenarios. The key advantages of *EyeRec* are:

- **Research Speed-up:** the code is freely available. Users can easily replace built-in algorithms to prototype their own algorithms or use the software as is for data acquisition.
- **Multiple Head-mounted Eye Tracking Hardware:** tested with several commercial eye trackers and USB cameras. Any eye tracker that exposes

Santini, T., Fuhl, W., Kübler, T. and Kasneci, E.
EyeRec: An Open-source Data Acquisition Software for Head-mounted Eye-tracking.
DOI: 10.5220/0005758903840389
In *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016) - Volume 3: VISAPP*, pages 386-391
ISBN: 978-989-758-175-5

its cameras via USB should work out of the box.

- **Real-time:** a low latency software pipeline allows its usage in time-critical applications.
- **State-of-the-art Pupil Detection:** ElSe (Fuhl et al., 2016), the top performer pupil detection algorithm, is fully integrated. Moreover, to demonstrate the ease of introducing new algorithms to *EyeRec*, we have integrated the popular Starburst (Li et al., 2005), Świrski et al. (Świrski et al., 2012), and ExCuSe (Fuhl et al., 2015) algorithms.
- **Data Streaming:** non-video data is streamed in real time through UDP/TCP connections, allowing easy integration with external buses (e.g., CAN).
- **Full Open-source Software Stack:** *EyeRec* fills the data acquisition gap. Combined with open-source eye-tracking data analysis software, such as *EyeTrace* (Kübler et al., 2015), a full open-source software stack is available.

The remainder of the paper is divided as follows. Section 2 describes the overall design and performance of *EyeRec*. Section 3 and Section 4 give details on the built-in pupil detection and gaze estimation algorithms, respectively. Section 5 concludes this work and presents future work.

2 *EyeRec*

EyeRec is written in `C++` and makes extensive use of the OpenCV (Bradski et al., 2000) library for image processing, VideoMan (Martirena, Javier B., 2015) library for video input, and the Qt (Qt Project, 2015) framework for its Graphical User Interface (GUI). At the moment, its development is focused on the Windows version; nonetheless, porting to other platforms should be possible as all components are cross-platform.

EyeRec is designed to work with monocular head-mounted eye trackers. Its architecture is based on a master/slave paradigm, based on the idea that information is most relevant when a new eye image becomes available. Thus, the eye camera is defined as the master and the scene camera as a slave. It is worth noticing that this master/slave scheme can be extended to a binocular system by treating the second eye camera as a slave. Whenever the master frame grabber captures a new *eye image* and its associated *timestamp*, *EyeRec* fetches the latest *scene image* from the slave frame grabber and bundles the images and timestamp in a single entity hereby referred to as *entry*. Additionally, *EyeRec* can be configured to undistort the field image if necessary, and camera calibration is available. This completes the *image acquisition* stage.

Afterwards, the newly generated entry is sent to the *image processing* stage. At this stage, the pupil detection algorithm is applied, and the resulting estimated pupil position is used to estimate the gaze position. Both pupil and gaze position estimates as well as other useful information (e.g., pupil area) are added to the *entry*, which is then forwarded to the *data streaming*, *data journaling*, and *GUI update* stages.

At the data streaming stage, all non-frame data present in the entry are transmitted through a TCP or UDP server (based on user configuration). At the *data journaling* stage, non-frame data are appended to a *journal* file, and frame data are appended to the pertinent videos. At the *GUI update* stage, pupil and gaze position are overlaid on frame data, which are then displayed to the user (see Figure 1).

Each of the aforementioned stages runs in its own execution thread. Image acquisition, image processing, and data streaming stages are all considered time-critical and run with highest priority. Remaining stages run with regular priority.

Figure 1: A subject with high degree of myopia (9.00D SPH) and astigmatism (5.00D CYL) stares at the *Recording Start* button in the GUI after a five point calibration. A challenging situation due to the glasses black frame, thick lenses, reflections, and eye camera position (off axis and not centered relative to the subject's eye).

An additional benefit from *EyeRec*'s master/slave architecture is that the data rate is synchronized with the master image sensor. Thus, provided that the master sensor produces new images at a constant rate, *EyeRec* will generate data entries at a constant rate, including the most relevant (i.e., the latest available) slave image. In comparison, the data recorded with proprietary software during our tests exhibited a large variance in the intersample period length, which suggests that software buffers are used to synchronize the data. This can be seem in Figure 2 and has a large impact in derived data that depends on the timestamps

(e.g., eye velocity) – for instance, eye velocity estimates with the similar physical displacement displays velocity discrepancies up to a factor of four times during our tests when using the software vendor. As a result, algorithms that depend on this information, such as the Velocity Threshold Identification (Salvucci and Goldberg, 2000) for saccade/fixation classification, can have their performance impaired.

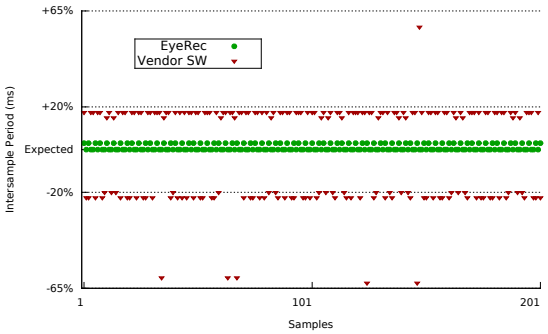

Figure 2: Period between adjacent samples produced by *EyeRec* and by proprietary software for the same eye tracker device. While the proprietary software exhibits large variance, *EyeRec*'s variance is only due to the rounding of the timestamp representation.

2.1 Data Format

Instead of reinventing the wheel, *EyeRec* follows the same data format as described in the *Dikablis Software Version 2.0 User Manual*. This makes possible to use data recorded with *EyeRec* in data analysis software that support Dikablis recordings. Upon recording, eye and scene videos are recorded, and data is stored in a journal file. Additionally, a file containing points used for calibration is also generated.

2.2 Supported Eye Trackers

Currently, the software supports Ergoneers' Dikablis Essential and Dikablis Professional eye trackers (Ergoneers, 2015). However, any eye tracker that exposes its cameras as USB cameras should work effortlessly. For instance, *EyeRec* is able to use regular web cameras instead of an eye tracker, although the built-in pupil-detection methods are heavily dependent on the quality of the eye image.

2.3 Pipeline Latency

We evaluated the latency of the software pipeline implemented in *EyeRec* using a Dikablis Professional eye tracker and a machine running Windows 8.1 with an Intel® Core™ i5-4590 @ 3.30GHz CPU and 8GB of RAM. Table 1 shows the latency of each stage relative to the moment a new eye image is available, so each consecutive step includes the processing time of prior steps. Data was collected during 10 minutes, yielding 18,000 samples. The eye tracker used for this evaluation provides 30 frames per second (for a single eye). Therefore, the deadline for the real-time stages is 33 ms, which is met with a slack larger than 20 ms.

Table 1: Latency relative to the moment a new eye image is available for each stage of the software pipeline (measured from 18,000 samples).

Stage	Mean Latency (ms)	Std. Dev.
Image Acquisition	1.36	0.32
Image Processing	9.16	0.83
Data Streaming	9.54	0.84
GUI Update	10.95	0.99
Data Journaling	17.25	2.07

2.4 Hardware Requirements

Given the slack for the reference machine, we estimate that any Intel® Core™ machine with at least 2GB of RAM should be able to handle the software requirements.

3 PUPIL DETECTION

Overall, we evaluated five candidates for the pupil detection algorithm.

Starburst (Li et al., 2005) first removes the corneal reflection and then selects pupil edge candidates along rays extending from a starting point. Returning rays are sent from the candidates found in the previous step, collecting additional candidates. This process is repeated iteratively using the average point from the candidates as starting point until convergence. Afterwards, inliers and outliers are identified using the RANSAC algorithm, a best fitting ellipse is determined, and the final ellipse parameters are determined by applying a model-based optimization.

Świrski et al. (Świrski et al., 2012) starts with Haar features of different sizes for coarse positioning. For a window surrounding the resulting position, an intensity histogram is calculated and clustered using the k-means algorithm. The separating intensity value between both clusters is used as threshold to extract the pupil. A modified RANSAC method is applied to the threshold-border.

SET (Javadi et al., 2015) first extracts pupil pixels based on a luminance threshold. The resulting image is then segmented, and the segment borders are extracted using the Convex Hull method. Ellipses are

fit to the segments based on their sinusoidal components, and the ellipse closest to a circle is selected as pupil.

ExCuSe (Fuhl et al., 2015) selects an initial method (best edge selection or coarse positioning) based on the intensity histogram of the input image. The best edge selection filters a Canny edge image based on morphologic operations and selects the edge with the darkest enclosed value. For the coarse positioning, the algorithm calculates the intersections of four orientations from the angular integral projection function (Mohammed et al., 2012). This coarse position is refined by analyzing the neighborhood of pixels in a window surrounding this position. The image is thresholded, and the border of threshold-regions is used additionally to filter a Canny edge image. The edge with the darkest enclosed intensity value is selected. For the pupil center estimation, a least squares ellipse fit is applied to the selected edge.

ElSe (Fuhl et al., 2016) applies a Canny edge detection method, removes edge connections that could impair the surrounding edge of the pupil, and evaluates remaining edges according to multiple heuristics to find a suitable pupil ellipse candidate. If the initial approach fails, the image is first downscaled. Then, its response to a surface difference filter is multiplied by its response to a mean filter, and the maximum response position is taken. This position is then refined on the upscaled image based on its pixel neighborhood. As ElSe is the default algorithm in *EyeRec* (see Section 3.1), its workflow is presented in more details in Figure 3.

3.1 Detection Rates

Table 2 shows the detection rates of all evaluated algorithms up to an Euclidean pixel distance of five to the hand-labeled pupil center. The data set is taken from (Fuhl et al., 2015; Świrski et al., 2012). All algorithms are used with the parameters provided by the authors. For Starburst, the starting location was set to the center of the image.

Table 2: Detection rate up to an error of five pixels of all integrated algorithms on all data sets proposed in (Świrski et al., 2012) and (Fuhl et al., 2015), totaling 39,001 images.

SET	Starburst	Świrski	ExCuSe	ElSe
14.0%	9.8%	24.8%	58.4%	74.05%

Based on these results, we chose ElSe as the default pupil detection algorithm. Nonetheless, Starburst, Świrski, and ExCuSe are also available to demonstrate *EyeRec's* modularity and how easily one can replace such essential parts.

4 GAZE ESTIMATION

The gaze estimation consists of a mapping of the pupil position in the eye image to the scene plane. This is typically done in two steps: calibration and projection. During calibration, a pupil position estimate is collected while the subject focuses on a known position in the scene image, forming a *calibration pair*.

4.1 Integrated Calibration

Typically, vendors use a fixed amount of calibration pairs (e.g., D-Lab always uses 4 points (Ergoneers, 2014)). *EyeRec* is unrestricted in this regard as it can collect as many calibration points as the user desires and, therefore, can refine calibration as well as report an estimate of calibration accuracy. Moreover, all collected pairs of pupil estimates and scene image coordinates are saved, making it possible to perform gaze estimation offline, e.g., removing individual calibration pairs, in case the online gaze estimation did not perform as desired.

4.2 Integrated Gaze Estimation

A gaze estimation based on the OpenCV *cv::findHomography* function is integrated in *EyeRec*. After a set of calibration pairs are collected, they are fed into the *cv::findHomography* function, yielding a calibration matrix. After successful calibration, each pupil position estimate is combined with the calibration matrix through the *cv::perspectiveTransform* function to generate a gaze position estimate. This procedure requires at least four calibration points.

5 FINAL REMARKS

In this paper, we introduced a new data acquisition software for head-mounted eye trackers. *EyeRec* has several key advantages over proprietary software (e.g., openness) and other open-source alternatives (e.g., multiple eye trackers support, improved pupil detection algorithm). In this way, *EyeRec* fills the data acquisition gap, enabling a full open-source software stack for for acquisition and analysis of eye-tracking data. The main shortcoming at the moment is the fact that the performance of the integrated gaze estimation method is not on par with state-of-the-art implementations, which usually consider device-specific effects. Besides solving the aforementioned shortcoming, future work includes automatic 3D eye model construction (Tsukada and

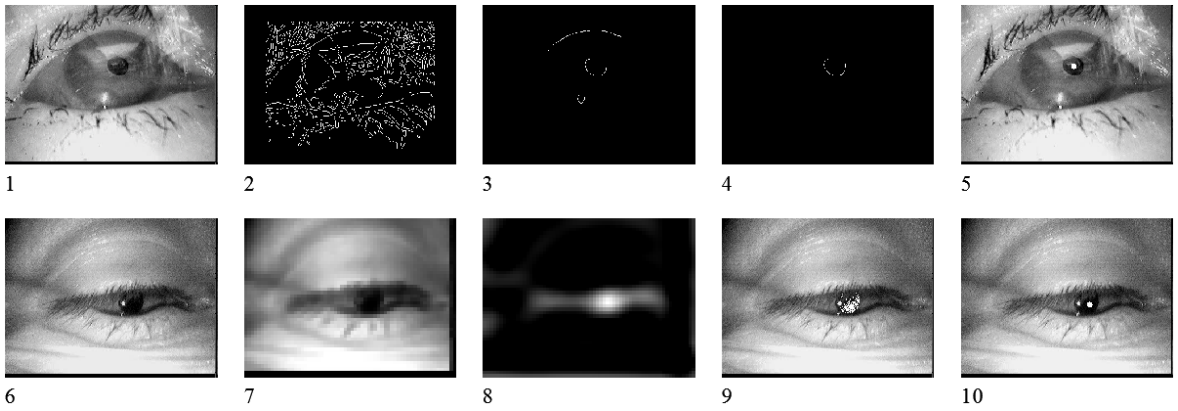

Figure 3: ElSe (Fuhl et al., 2016) workflow. (1) input image. (2) Canny edge image. (3) The remaining edges after curvature analysis, analysis of the enclosed intensity value, and shape. (4) The curvature with the most enclosing low intensity values and the most circular ellipse is chosen as the pupil boundary. (5) The resulting pupil center estimate. In case the initial estimate fails to produce a valid ellipse: the input image (6) is downscaled (7). (8) The downscaled image after convolution with a mean and a surface difference filters. (9) The threshold for pupil region extraction is calculated and used for pupil area estimation. (10) The resulting pupil center estimate.

Kanade, 2012; Świrski and Dodgson, 2013), support for remote gaze estimation (Model and Eizenman, 2010; Yamazoe et al., 2008), additional calibration methods (Guestrin and Eizenman, 2006; Pirri et al., 2011), real-time eye movement classification based on Bayesian mixture models (Kasneci et al., 2014; Kasneci et al., 2015; Santini et al., 2016), automatic blink detection, adding support for other eye trackers, and adding support for binocular systems as well as external, user definable, triggers events. Feedback from the community on future features is also welcome.

Source code, binaries for Windows, and extensive documentation are available at:
www.perception.uni-tuebingen.de

REFERENCES

Bradski, G. et al. (2000). The OpenCV Library. *Doctor Dobbs Journal*, 25(11):120–126.

Dalmaijer, E. S., Mathôt, S., and Van der Stigchel, S. (2014). Pygaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. *Behavior research methods*, 46(4).

Duchowski, A. T. (2002). A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments, & Computers*, 34(4):455–470.

Ergoneers (2014). *D-Lab Manual*.

Ergoneers (2015). Dikablis Glasses. http://www.ergoneers.com/en/hardware/eye-tracking/.

Fuhl, W., Kübler, T., Sippel, K., Rosenstiel, W., and Kasneci, E. (2015). Excuse: Robust pupil detection in real-world scenarios. In *Computer Analysis of Images and Patterns 2015. CAIP 2015. 16th International Conference*. IEEE.

Fuhl, W., Santini, T., Kübler, T., and Kasneci, E. (2016). Else: Ellipse selection for robust pupil detection in real-world environments. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM. Forthcoming.

Guestrin, E. D. and Eizenman, M. (2006). General theory of remote gaze estimation using the pupil center and corneal reflections. *Biomedical Engineering, IEEE Transactions on*, 53(6):1124–1133.

Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., and Van de Weijer, J. (2011). *Eye tracking: A comprehensive guide to methods and measures*. Oxford University Press.

Holsanova, J., Rahm, H., and Holmqvist, K. (2006). Entry points and reading paths on newspaper spreads: comparing a semiotic analysis with eye-tracking measurements. *Visual communication*, 5(1):65–93.

Javadi, A.-H., Hakimi, Z., Barati, M., Walsh, V., and Tcheang, L. (2015). Set: a pupil detection method using sinusoidal approximation. *Frontiers in neuroengineering*, 8.

Kasneci, E., Kasneci, G., Kübler, T. C., and Rosenstiel, W. (2014). The applicability of probabilistic methods to the online recognition of fixations and saccades in dynamic scenes. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '14, pages 323–326, New York, NY, USA. ACM.

Kasneci, E., Kasneci, G., Kübler, T. C., and Rosenstiel, W. (2015). Online recognition of fixations, saccades, and smooth pursuits for automated analysis of traffic hazard perception. In Koprinkova-Hristova, P., Mladenov, V., and Kasabov, N. K., editors, *Artificial Neural Networks*, volume 4 of *Springer Series in Bio-/Neuroinformatics*, pages 411–434. Springer International Publishing.

Kassner, M., Patera, W., and Bulling, A. (2014). Pupil: An

open source platform for pervasive eye tracking and mobile gaze-based interaction. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, pages 1151–1160, New York, NY, USA. ACM.

Kübler, T., Sippel, K., Fuhl, W., Schievelbein, G., Aufreiter, J., Rosenberg, R., Rosenstiel, W., and Kasneci, E. (2015). *Analysis of eye movements with Eyetrace*, volume In press. Communications in Computer and Information Science (CCIS), Biomedical Engineering Systems and Technologies. Springer International Publishing.

Li, D., Babcock, J., and Parkhurst, D. J. (2006). openeyes: A low-cost head-mounted eye-tracking solution. In *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*, ETRA '06, pages 95–100, New York, NY, USA. ACM.

Li, D., Winfield, D., and Parkhurst, D. J. (2005). Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *Computer Vision and Pattern Recognition Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, pages 79–79. IEEE.

Martirena, Javier B. (2015). The VideoMan Library. http://videomanlib.sourceforge.net/.

Model, D. and Eizenman, M. (2010). User-calibration-free remote gaze estimation system. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, pages 29–36. ACM.

Mohammed, G. J., Hong, B. R., and Jarjes, A. A. (2012). Accurate pupil features extraction based on new projection function. *Computing and Informatics*, 29(4):663–680.

Pirri, F., Pizzoli, M., and Rudi, A. (2011). A general method for the point of regard estimation in 3d space. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 921–928. IEEE.

Qt Project (2015). The Qt Framework. http://www.qt.io/.

Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological bulletin*, 124(3):372.

Salvucci, D. D. and Goldberg, J. H. (2000). Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications*, pages 71–78. ACM.

Santini, T., Fuhl, W., Kübler, T., and Kasneci, E. (2016). Bayesian identification of fixations, saccades, and smooth pursuits. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM. Forthcoming.

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

Świrski, L. and Dodgson, N. A. (2013). A fully-automatic, temporal approach to single camera, glint-free 3d eye model fitting. In *Proceedings of ECEM 2013*.

Tsukada, A. and Kanade, T. (2012). Automatic acquisition of a 3d eye model for a wearable first-person vision device. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 213–216. ACM.

Yamazoe, H., Utsumi, A., Yonezawa, T., and Abe, S. (2008). Remote gaze estimation with a single camera based on facial-feature tracking without special calibration actions. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, pages 245–250. ACM.