EyeRecToo: Open-source Software for Real-time Pervasive Head-mounted Eye Tracking

Thiago Santini, Wolfgang Fuhl, David Geisler and Enkelejda Kasneci
Perception Engineering, University of Tübingen, Tübingen, Germany
{*thiago.santini, wolfgang.fuhl, david.geisler, enkelejda.kasneci*}*@uni-tuebingen.de*

Keywords: Eye Movements, Pupil Detection, Calibration, Gaze Estimation, Open-source, Eye Tracking, Data Acquisition, Human-computer Interaction, Real-time, Pervasive.

Abstract: Head-mounted eye tracking offers remarkable opportunities for research and applications regarding pervasive health monitoring, mental state inference, and human computer interaction in dynamic scenarios. Although a plethora of software for the acquisition of eye-tracking data exists, they often exhibit critical issues when pervasive eye tracking is considered, e.g., closed source, costly eye tracker hardware dependencies, and requiring a human supervisor for calibration. In this paper, we introduce *EyeRecToo*, an open-source software for real-time pervasive head-mounted eye-tracking. Out of the box, *EyeRecToo* offers multiple real-time state-of-the-art pupil detection and gaze estimation methods, which can be easily replaced by user implemented algorithms if desired. A novel calibration method that allows users to calibrate the system without the assistance of a human supervisor is also integrated. 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, *EyeRecToo* serves as a framework to quickly enable pervasive eye-tracking research and applications. Available at: www.ti.uni-tuebingen.de/perception.

1 INTRODUCTION

In the past two decades, the number of researchers using eye trackers has grown enormously (Holmqvist et al., 2011), with researchers stemming from several distinct fields (Duchowski, 2002). For instance, eye tracking has been employed from simple and fixed research scenarios – e.g., language reading (Holmqvist et al., 2011) – to complex and dynamic cases – e.g., driving (Kasneci et al., 2014) and ancillary operating microscope controls (Fuhl et al., 2016b). In particular, pervasive eye tracking also has the potential for health monitoring (Vidal et al., 2012), mental state inference (Fuhl et al., 2016b), and human computer interaction (Majaranta and Bulling, 2014). Naturally, these distinct use cases have specific needs, leading to the spawning of several 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. However, existing eye tracking systems often present multiple critical issues when pervasive eye tracking is considered. For instance, commercial systems rely on closed-source software, offering their eye tracker bundled with their own software solutions. Besides the high costs involved, researchers and application developers have practically no direct alternatives if the system does not work under the required conditions (Santini et al., 2016b). Other (open-source) systems – e.g., openEyes (Li et al., 2006a), PyGaze (Dalmaijer et al., 2014), Pupil (Pupil Labs, 2016), and EyeRec (Santini et al., 2016b) – either focus on their own eye trackers, depend on existing APIs from manufacturers, or require a human supervisor in order to calibrate the system.

In this paper, we introduce *EyeRecToo*[1], an open-source software for pervasive head-mounted eye tracking that solves all of the aforementioned issues to quickly enable pervasive eye tracking research and applications; its key advantages are as follow.

- **Open and Free:** 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 and human-computer interfaces.
- **Data Streaming:** non-video data is streamed in

[1]The name is a wordplay on the competitor EyeRec "I Rec[ord]" since the software provides similar recording functionality; hence, *EyeRecToo* "I Rec[ord] Too". Permission to use this name was granted by the EyeRec developers.

Santini T., Fuhl W., Geisler D. and Kasneci E.
EyeRecToo: Open-source Software for Real-time Pervasive Head-mounted Eye Tracking.
DOI: 10.5220/0006224700960101
In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017)*, pages 96-101
ISBN: 978-989-758-227-1

real time through a UDP stream, allowing easy and quick integration with external buses – e.g., automotive CAN – or user applications.

- **Hardware Independency:** the software handles from cheap homemade head-mounted eye trackers assembled from regular webcams to expensive commercial hardware[2].
- **Real-time:** a low latency software pipeline enables its usage in time-critical applications.
- **State-of-the-art Pupil Detection:** *ElSe*, the top performer pupil detection algorithm, is fully integrated. Additional integrated methods include *Starburst*, *Świrski*, and *ExCuSe*.
- **State-of-the-art Gaze Estimation:** parameterizable gaze estimation based on polynomial regression (Cerrolaza et al., 2012) and homography (Yu and Eizenman, 2004).
- **Unsupervised Calibration:** a method that allows the user to quickly calibrate the system, independently from assistance from a second individual.
- **Complete Free Software Stack:** combined with free eye-tracking data analysis software, such as *EyeTrace* (Kübler et al.,), a full eye tracking software stack is accessible free of cost.

2 EyeRecToo

EyeRecToo is written in `C++` and makes extensive use of the *OpenCV 3.1.0* (Bradski et al., 2000) library for image processing and the *Qt 5.7* framework (Qt Project, 2016) for its multimedia access and Graphical User Interface (GUI). Development and testing are focused on a *Windows* platform; however the software also runs under *Linux*[3]. Moreover, porting to other platforms (e.g., *Android*, *OS X*) should be possible as all components are cross-platform.

EyeRecToo is built around widgets that provide functionality and configurability to the system. It was designed to work with a field camera plus mono or binocular head-mounted eye trackers but also foresees the existence of other data input devices in the future – e.g., an Inertial Measurement Unit (IMU) for head-movement estimation (Larsson et al., 2016). In fact, *EyeRecToo* assumes there is no hardware synchronization between input devices and has a built-in software *synchronizer*. Each input device is associated with an *input widget*. *Input widgets* register with the *synchronizer*, read data from the associated device, timestamp the incoming data according to a global monotonic reference clock, possibly process the data to extend it (e.g., pupil detection), and save the resulting extended data, which is also sent to the synchronizer. The *synchronizer*'s task is then to store the latest data incoming from the *input widgets* (according to a time window specified by the user) and, at predefined intervals, generate a *DataTuple* with timestamp t containing the data from each registered *input widget* with timestamp closest in time to t, thus synchronizing the input devices data[4]. The resulting *DataTuple* is then forwarded to the *calibration/gaze estimation*, which complements the tuple with gaze data. The complete tuple is then stored (*data journaling*), broadcasted through UDP (*data streaming*), and exhibited to the user (*GUI update*). This results in a modular and easily extensible design that allows one to reconstruct events as they occurred during run time.

2.1 Input Widgets

Currently two input widgets are implemented in the system: the eye camera input widget, and the field camera input widget. These run in individual threads with highest priority in the system. The eye camera input widget is designed to receive close-up eye images, allowing the user to select during run time the input device, region of interest (ROI) in which eye feature detection is performed, image flipping, and pupil detection algorithm. Available pupil detection algorithms and their performance are described in detail in Section 3. The field camera input widget is designed to capture images from the point of view (POV) of the eye tracker wearer, allowing the user to select during run time the input device, image undistortion, image flipping and fiducial marker detection. Additionally, camera intrinsic and extrinsic parameter estimation is built-in. Currently, ArUcO (Garrido-Jurado et al., 2014) markers are supported. Both widgets store the video (as fed to their respective image processing algorithms) as well as their respective detection algorithm data and can be use independently from other parts of the system (e.g., to detect pupils in eye videos in an offline fashion).

2.2 Calibration / Gaze Estimation Widget

This widget provides advanced calibration and gaze estimation methods, including two methods for cal-

[2]Provided cameras are accessible through DirectShow (Windows) or v4l2 (Linux).

[3]Windows 8.1/MSVC2015 and Ubuntu 16.04/gcc 5.4.0.

[4]In case the input devices are hardware-synchronized, one can use a delayed trigger based on any of the input devices to preserve synchronization.

ibration (*supervised* / *unsupervised*). The *supervised* calibration is a typical eye tracker calibration, which requires a human supervisor to coordinate with the user and select points of regard that the user gazes during calibration. The *unsupervised* calibration methods as well as available gaze estimation methods are described in depth in Section 4. *EyeRecToo* is also able to automatically reserve some of the calibration points for a less biased evaluation of the gaze estimation function. Moreover, functionalities to save and load data tuples (both for calibration and evaluation) are implemented, which allows developers to easily prototype new calibration and gaze estimation methods based on existing data.

2.3 Supported Eye Trackers

Currently, the software supports Ergoneers' Dikablis Essential and Dikablis Professional eye trackers (Ergoneers, 2016), Pupil Do-It-Yourself kit (Pupil Labs, 2016), and the PS3Eye-based operating microscope add-on module proposed by Eivazi et al. (Eivazi et al., 2016)[5]. However, any eye tracker that provides access to its cameras through *DirectShow* (on Windows) or *v4l2* (on Linux) should work effortlessly. For instance, *EyeRecToo* 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 – in particular, pupil detection methods are often designed for near-infrared images.

2.4 Software Latency and Hardware Requirements

We evaluated the latency of the software pipeline implemented in *EyeRecToo* using the default configuration, 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. In total, 30.000 samples were collected from each input device. Table 1 shows the latency of operations that require a significant amount of time relative to the intersample period of the eye tracker fastest input device (16.67 ms), namely image acquisition/processing and storage[6]. It is worth noticing that, given enough available processing cores, these operations can be realized in a parallel fashion; thus, the remaining slack based on the 16.67 ms deadline is ≈ 8 ms. Given these measurements, we estimate that any Intel® Core™ machine with four cores and 2GB of RAM should be able to meet the software requirements.

[5]Provided that the module remain static w.r.t. the head.

[6]Values are based on the default pupil detection (*ElSe*) and field images containing at least four markers.

Table 1: Resulting latency (mean±standard deviation) for time-consuming operations from the 30.000 samples of each available input widgets.

Operation	Input Widget	Latency (ms)
Processing	Eye Camera	8.35 ± 0.73
	Field Camera	4.97 ± 1.16
Storage	Eye Camera	2.85 ± 1.23
	Field Camera	4.39 ± 1.74

3 PUPIL DETECTION

EyeRecToo offers four integrated pupil detection algorithms, which were chosen based on their detection rate performance – e.g., *ElSe* (Fuhl et al., 2016a) and *ExCuSe* (Fuhl et al., 2015) – and popularity – e.g., *Świrski* (Świrski et al., 2012) and *Starburst* (Li et al., 2005). Since *EyeRecToo*'s goal is to enable pervasive eye tracking, the main requirements for pupil detection algorithms are real-time capabilities and high detection rates on challenging and dynamic scenarios. Based on these requirements, *ElSe* (Fuhl et al., 2016a) was selected as default pupil detection algorithm; the resulting detection rate performance of these algorithms on the data sets provided by (Fuhl et al., 2016a; Fuhl et al., 2015; Świrski et al., 2012) is shown in Figure 1. A brief description of each algorithm follows; a detailed review of these algorithms is given in (Fuhl et al., 2016c).

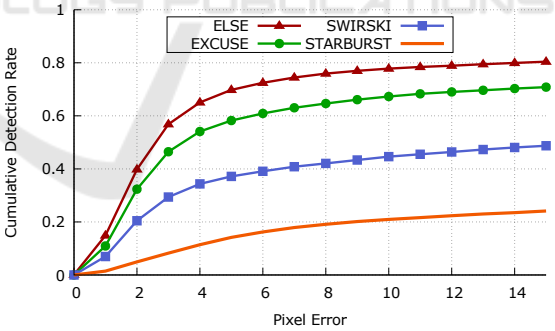

Figure 1: Cumulative detection rate given the distance between detected pupil position relative to a human-annotated ground-truth distance for each of the available algorithms based on the data from (Fuhl et al., 2016a).

ElSe (Fuhl et al., 2016a) 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 this initial approach fails, the image is downscaled, and a second approach attempted. The downscaled image's response to a surface difference filter is multiplied by the complement of its mean filter response, and the

maximum of the resulting map is taken. This position is then refined on the upscaled image based on its pixel neighborhood.

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. 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.

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 thresholded pupil border.

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.

4 CALIBRATION AND GAZE ESTIMATION

In video-based eye tracking, gaze estimation is the process of estimating the Point Of Regard (POR) of the user given eye images. High-end state-of-the-art mobile eye tracker systems (e.g., SMI and Tobii glasses (SensoMotoric Instruments GmbH, 2016; Tobii Technology, 2016)) rely on geometry-based gaze estimation approaches, which can provide gaze estimations without calibration. In practice, it is common to have at least an one point calibration to adapt the geometrical model to the user and estimate the angle between visual and optical axis, and it has been reported that additional points are generally required for good accuracy (Villanueva and Cabeza, 2008). Furthermore, such approaches require specialized hardware (e.g., multiple cameras and glint points), cost in the order of tens of thousands of $USD, and are susceptible to inaccuracies stemming from lens distortions (Kübler et al., 2016). On the other hand, mobile eye trackers that make use of regression-based gaze-mappings require a calibration step but automatically adapt to distortions and are comparatively low-cost – e.g., a research grade binocular eye tracker from Pupil Labs is available for $2340 EUR (Pupil Labs, 2016). Moreover, similar eye trackers have been demonstrated by mounting two (an eye and a field) cameras onto the frames of glasses (Babcock and Pelz, 2004; Li et al., 2006b; San Agustin et al., 2010), yielding even cheaper alternatives for the more tech-savy users. Therefore, we focus on regression based alternatives, which require calibration.

4.1 Calibration

In its current state, the calibration step presents some disadvantages and has been pointed out as one of the main factors hindering a wider adoption of eye tracking technologies (Morimoto and Mimica, 2005). Common calibration procedures customarily require the assistance of an individual other than the eye tracker user in order to calibrate (and check the accuracy of) the system. The user and the aide must coordinate so that the aide selects calibration points accordingly to the user's gaze. As a result, current calibration procedures cannot be performed individually and require a considerable amount of time to collect even a small amount of calibration points, impeding their usage for ubiquitous eye tracking. *EyeRecToo* provides functionality for these *supervised* calibrations such that the users can collect as many eye-gaze relationships as necessary as well as choose to sample a single median point or multiple points per relationship. Furthermore, audible cues are also provided to minimize the amount of interaction between user and supervisor, thus diminishing human error and calibration time.

Besides the regular *supervised* calibration, *EyeRecToo* integrates a novel *unsupervised* approach that enables users to quickly and independently calibrate head-mounted eye trackers by gazing at a fiducial marker that moves w.r.t. the user's head. Additionally, we also provide a companion *Android* application that can be used to display the marker and receive feedback regarding the quality of the calibration (see Figure 2). Alternatively, a user can also perform this calibration using a printed marker or by dis-

playing the marker on any screen. After collecting several calibration points, *EyeRecToo* then removes inferior and wrong eye-gaze relationships according to a series of rationalized approaches based on domain specific assumptions regarding head-mounted eye tracking setups, data, and algorithms (see Figure 3). From the remaining points, some are reserved for evaluation based on their spatial location, and the remaining points are used for calibration. This calibration method, dubbed *CalibMe*, is described in detail in (Santini et al., 2017).

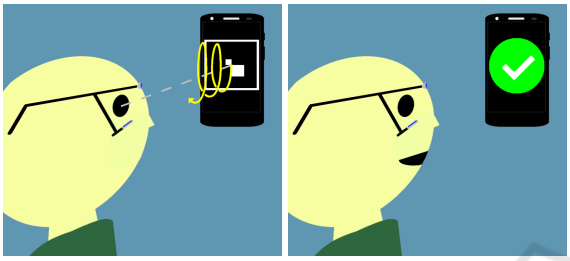

Figure 2: While gazing at the center of the marker, the user moves the smartphone displaying the collection marker to collect eye-gaze relationships (left). The eye tracker then notifies the user that the calibration has been performed successfully through visual/haptic/audible feedback, signaling that the system is now ready to use (right).

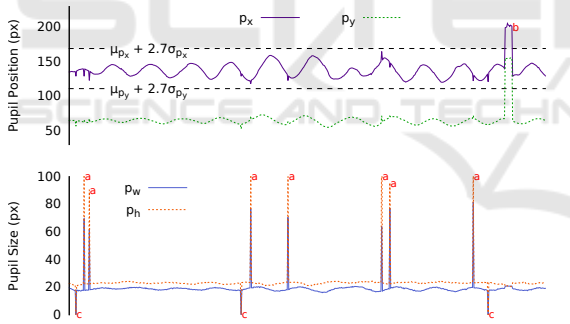

Figure 3: Rationalized outlier removal. Outliers based on subsequent pupil size (a), pupil position range (b), and pupil detection algorithm specific information (c). Notice how the pupil position estimate (p_x, p_y) is corrupted by such outliers.

4.2 Gaze Estimation

These two available calibration methods are complemented by multiple gaze estimation methods. Out of the box, six polynomial regression approaches are offered – ranging from first to third order bivariate polynomial least-square fits through single value decomposition. Furthermore, an additional approach based on projective geometry is also available through projective space isomorphism (i.e., homography). It is worth noticing however that the latter requires that camera parameters be taken into account[7].

To provide gaze estimation accuracy figures, we conducted an evaluation employing a second order bivariate regression with five adult subjects (4 male, 1 female) – two of which wore glasses during the experiments. The experiment was conducted using a Dikablis Pro eye tracker (Ergoneers, 2016). This device has two eye (@60 Hz) and one field (@30 Hz) cameras; data tuples were sampled based on the frame rate of the field camera. These experiments yielded a mean angular error averaged over all participants of 0.59° ($\sigma = 0.23^\circ$) when calibrated with the *unsupervised* method. In contrast, a regular *supervised* nine points calibration yielded a mean angular error of 0.82° ($\sigma = 0.15^\circ$). Both calibrations exhibited accuracies well within physiological values, thus attesting for the efficacy of the system as a whole.

5 FINAL REMARKS

In this paper, we introduced a software for pervasive and real-time head-mounted eye trackers. *EyeRecToo* 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, unsupervised calibration). Future work includes automatic 3D eye model construction (Świrski and Dodgson, 2013), support for remote gaze estimation (Model and Eizenman, 2010), additional calibration methods (Guestrin and Eizenman, 2006), real-time eye movement classification based on Bayesian mixture models (Kasneci et al., 2015; Santini et al., 2016a), automatic blink detection (Appel et al., 2016), and support for additional eye trackers.

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

REFERENCES

Appel, T. et al. (2016). Brightness- and motion-based blink detection for head-mounted eye trackers. In *Proc. of the Int. Joint Conf. on Pervasive and Ubiquitous Computing, UbiComp Adjunct*. ACM.

Babcock, J. S. and Pelz, J. B. (2004). Building a lightweight eyetracking headgear. In *Proc. of the 2004 Symp. on Eye tracking research & applications*. ACM.

Bradski, G. et al. (2000). The OpenCV Library. *Doctor Dobbs Journal*.

[7] *EyeRecToo* has built-in methods to estimate the intrinsic and extrinsic camera parameters if necessary.

Cerrolaza, J. J. et al. (2012). Study of polynomial mapping functions in video-oculography eye trackers. *Trans. on Computer-Human Interaction (TOCHI)*.

Dalmaijer, E. S. et al. (2014). Pygaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. *Behavior research methods*.

Duchowski, A. T. (2002). A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments, & Computers*.

Eivazi, S. et al. (2016). Embedding an eye tracker into a surgical microscope: Requirements, design, and implementation. *IEEE Sensors Journal*.

Ergoneers (2016). Dikablis. www.ergoneers.com.

Fuhl, W. et al. (2015). Excuse: Robust pupil detection in real-world scenarios. In *Computer Analysis of Images and Patterns 2015. CAIP 2015. 16th Int. Conf.* IEEE.

Fuhl, W. et al. (2016a). Else: Ellipse selection for robust pupil detection in real-world environments. In *Proc. of the Symp. on Eye Tracking Research & Applications*. ACM.

Fuhl, W. et al. (2016b). Non-intrusive practitioner pupil detection for unmodified microscope oculars. *Computers in Biology and Medicine*.

Fuhl, W. et al. (2016c). Pupil detection for head-mounted eye tracking in the wild: an evaluation of the state of the art. *Machine Vision and Applications*.

Garrido-Jurado, S. et al. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*.

Guestrin, E. D. and Eizenman, M. (2006). General theory of remote gaze estimation using the pupil center and corneal reflections. *Biomedical Engineering, IEEE Trans. on*.

Holmqvist, K. et al. (2011). *Eye tracking: A comprehensive guide to methods and measures*. Oxford University.

Kasneci, E. et al. (2014). The applicability of probabilistic methods to the online recognition of fixations and saccades in dynamic scenes. In *Proc. of the Symp. on Eye Tracking Research and Applications*.

Kasneci, E. et al. (2015). Online recognition of fixations, saccades, and smooth pursuits for automated analysis of traffic hazard perception. In *Artificial Neural Networks*. Springer.

Kübler, T. C. et al. Analysis of eye movements with eyetrace. In *Biomedical Engineering Systems and Technologies*. Springer.

Kübler, T. C. et al. (2016). Rendering refraction and reflection of eyeglasses for synthetic eye tracker images. In *Proc. of the Symp. on Eye Tracking Research & Applications*. ACM.

Larsson, L. et al. (2016). Head movement compensation and multi-modal event detection in eye-tracking data for unconstrained head movements. *Journal of Neuroscience Methods*.

Li, D. et al. (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 Conf. on*. IEEE.

Li, D. et al. (2006a). openeyes: A low-cost head-mounted eye-tracking solution. In *Proc. of the Symp. on Eye Tracking Research &Amp; Applications*.

Li, D. et al. (2006b). openeyes: a low-cost head-mounted eye-tracking solution. In *Proc. of the 2006 Symp. on Eye tracking research & applications*. ACM.

Majaranta, P. and Bulling, A. (2014). *Eye Tracking and Eye-Based Human-Computer Interaction*. Advances in Physiological Computing. Springer.

Model, D. and Eizenman, M. (2010). User-calibration-free remote gaze estimation system. In *Proc. of the Symp. on Eye-Tracking Research & Applications*. ACM.

Morimoto, C. H. and Mimica, M. R. (2005). Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*.

Pupil Labs (2016). www.pupil-labs.com/. Accessed: 16-09-07.

Qt Project (2016). Qt Framework. www.qt.io/.

San Agustin, J. et al. (2010). Evaluation of a low-cost open-source gaze tracker. In *Proc. of the 2010 Symp. on Eye-Tracking Research & Applications*. ACM.

Santini, T. et al. (2016a). Bayesian identification of fixations, saccades, and smooth pursuits. In *Proc. of the Symp. on Eye Tracking Research & Applications*. ACM.

Santini, T. et al. (2016b). Eyerec: An open-source data acquisition software for head-mounted eye-tracking. In *Proc. of the Joint Conf. on Computer Vision, Imaging and Computer Graphics Theory and Applications*.

Santini, T. et al. (2017). CalibMe: Fast and unsupervised eye tracker calibration for gaze-based pervasive human-computer interaction. In *Proc. of the CHI Conf. on Human Factors in Computing Systems*.

SensoMotoric Instruments GmbH (2016). www.smivision.com/. Accessed: 16-09-07.

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

Świrski, L. et al. (2012). Robust real-time pupil tracking in highly off-axis images. In *Proc. of the Symp. on Eye Tracking Research and Applications*. ACM.

Tobii Technology (2016). www.tobii.com. Accessed: 16-09-07.

Vidal, M. et al. (2012). Wearable eye tracking for mental health monitoring. *Computer Communications*.

Villanueva, A. and Cabeza, R. (2008). A novel gaze estimation system with one calibration point. *IEEE Trans. on Systems, Man, and Cybernetics*.

Yu, L. H. and Eizenman, M. (2004). A new methodology for determining point-of-gaze in head-mounted eye tracking systems. *IEEE Trans. on Biomedical Engineering*.