Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

ETCAL - A versatile and extendable library for eye tracker calibration

Pawel Kasprowski [*], Katarzyna Harezlak

Institute of Informatics, Silesian University of Technology, ul. Akademicka 16, 44-100 Gliwice, Poland

A R T I C L E   I N F O

Article history:
Received 30 November 2017
Accepted 11 December 2017

Keywords:
Eye tracking
Calibration
Library

A B S T R A C T

Recently eye tracking has become a popular technique that may be used for variety of applications starting from medical ones, through psychological, analyzing user experience, ending with interactive games. Video based oculography (VOG) is the most popular technique because it is non-intrusive, can be use in users' natural environment and are relatively cheap as it uses only classic cameras. There are already well established methods for eye detection on a camera capture. However, to be usable in gaze position estimation, this information must be associated with an area in an observer scene, which requires evaluating several parameters. These parameters are typically estimated during the process called *calibration*. The main purpose of the software described in this paper is to establish a common platform that is easy to use and may be used in different calibration scenarios. Apart from the normal regression based calibration the ETCAL library allows also to use more sophisticated methods like automatic parameters optimization or the automatic detection of gaze targets. The library is also easily extendable and may be accessed with a convenient Web/REST interface.

Code metadata

Current code version	0.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-17-00081
Legal Code License	GNU LGPL
Code versioning system used	git
Software code languages, tools, and services used	Java
Compilation requirements, operating environments & dependencies	Maven project, may be compiled as a library (jar) or as a web application (war)
If available Link to developer documentation/manual	https://github.com/kasprowski/etcal/doc
Support email for questions	pawel@kasprowski.pl

Software metadata

Current software version	0.1
Permanent link to executables of this version	https://github.com/kasprowski/etcal
Legal Software License	GNU LGPL
Computing platforms/Operating Systems	Required a platform with Java Virtual Machine 1.8 or higher, a web version may be started on any servlet container (Tomcat, Glassfish, WildFly etc.)
Installation requirements & dependencies	all dependencies in a pom.xml Maven file
If available, link to user manual – if formally published include a reference to the publication in the reference list	https://github.com/kasprowski/etcal/doc
Support email for questions	pawel@kasprowski.pl

1. Motivation and significance

There are plenty of potential applications of eye tracking: medicine [1,2], psychology [3], sociology [4], education [5], usability assessment [6] or advertisement analysis. There is also a growing number of applications using eye gaze as a new input

DOI of original article: http://dx.doi.org/10.1016/j.dsp.2017.11.011.

[*] Corresponding author.

E-mail address: kasprowski@polsl.pl (P. Kasprowski).

https://doi.org/10.1016/j.softx.2017.12.005
2352-7110/© 2017 Published by Elsevier B.V.

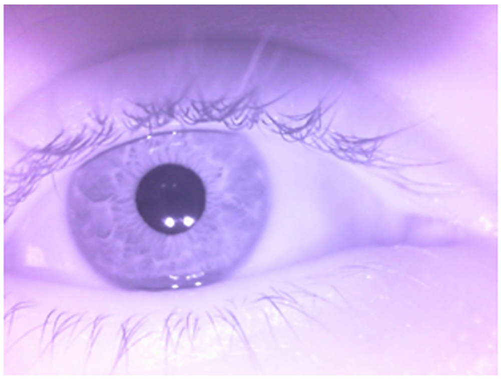


Fig. 1. A typical eye image used by eye tracker camera.

modality [7]. With access to high quality cameras and devices that are able to process sophisticated image retrieval tasks, anybody can build an eye tracker. Many algorithms are able to find an eye center, a glint (light reflection) or eye corners in a provided image (Fig. 1). However, these values must be mapped to coordinates related to a scene that a user is looking at. This transformation is necessary to make reasoning about peoples' points of interest.

The solution, which has to be applied is called a *calibration* and must precede all eye movement registration. There are several types of stimuli, which may be used during the calibration phase. The most popular is a jumping point – a dot appearing on a screen in different locations – which is commonly used by eye tracker manufacturers. Participants are expected to follow this point with eyes. The other possibility is the usage of a wandering point, which smoothly changes its position on a screen causing the effect of smooth pursuit eye movement [8,9]. More complicated stimuli, in form of images, may be used as well, however in such a case well-defined targets are required. There are studies, in which the calibration process is included in normal human–computer interaction [10]. Especially mouse click positions may be utilized as probable gaze locations [11]. Such an environment may provide more than one possible target to gaze at, thus the special algorithm, able to choose probable locations (targets) to obtain the best possible model is required.

Manufacturers of eye trackers, together with their devices, provide a proprietary software, however, it often suffers from limited calibration scenarios and it is not clear how it works so it is difficult to tune it. Application of more sophisticated and comparable solutions requires a new piece of software to be prepared. The same need arises in the case of self-developed eye trackers [12,13].

The motivation of this work was to provide easy to use and available online tool capable to calculate gaze points based on raw eye movement recordings and yield reproducible calibration results. It may be useful for researchers utilizing an untypical calibration scenario or producing their own eye trackers. The tool may also be used as the common platform to compare results obtained for data collected with the usage of different eye trackers. Additionally, the ETCAL library contains more sophisticated heuristic algorithms that enable automatic optimization of various calibration models (like polynomial or SVR) using parametrizable cross validation and automatic detection of gaze targets in the case of implicit calibration scenarios.

2. Software description

The library is the comprehensive tool that enables preparation of calibration models. It is written in Java and the whole code is available on github repository thus possible to clone or fork.

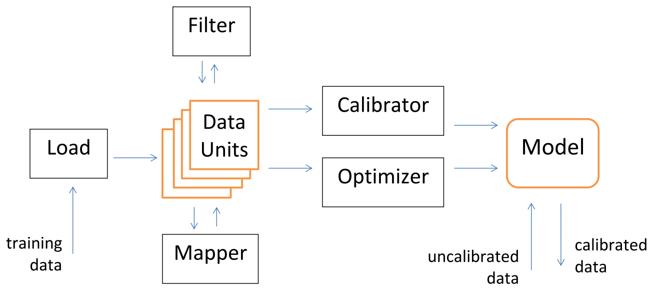


Fig. 2. The ETCAL library architecture.

There are two ways as to how to use the ETCAL library: via its programming interface or via its REST interface. While the former is convenient for Java programmers as all they need is to add the library's jar file with dependencies to their projects, the latter may be used by any software that is able to produce HTTP requests. What is very important, nearly every ETCAL procedure has a synchronic and asynchronic versions. It enables prospective users to start a long lasting process and continue their own work without a necessity to idly wait for the results. While the ETCAL is equipped with several ready-to-use algorithms, it may be easily extended. Interfaces that should be implemented are thoroughly explained in the ETCAL's documentation.

Basically, the first step when using the ETCAL is to create an *EtCal* object and upload the training data. The training data consist of eye tracker output and predicted gaze coordinates on some plane (*Targets* objects). After the training data is loaded there are six main activities that may be executed (Fig. 2):

1. filter — The data may be filtered using one of the predefined or any newly self-implemented *Filter*.
2. map — If the data consists of data units for which there are more than one possible targets, it is possible to start a *Mapper* that uses a heuristic algorithm to reversibly find the correct target from a list of possibilities for the given data unit.
3. build model — The training data may be used to build the calibration model using one of the predefined or own *Calibrator*.
4. optimize — The data may be used to tune calibrator's parameters to give the lowest errors for the data. There may be different *Optimizers* used with different calibrator types, cross validation types and optimization algorithms.
5. calibrate — After a model has been built either directly by a calibrator or by an optimizer (that also uses a calibrator but under the surface) it is possible to use it to calibrate a new data. For a provided set of features the ETCAL returns a Target object with gaze coordinates.
6. checkErrors, plotErrors — It is also possible to provide a new dataset, calibrate it and check errors (R^2, *RMSE*) or get an image which illustrates the errors on a plot.

2.1. Input data format

The input dataset consists of a sequence of data units. Every data unit consists of the set of variables (V) and corresponding targets T (gaze coordinates):

```
{"variables":[20.0,13.0,10.0,20.0],
"targets":[{"x":260.0,
"y":260.0,"w":1.0},{"x":360.0,"y":360.0,"w":1.0}]}
```

Every variable must be a floating point number. As it is visible on the listing there may be more than one targets defined for one data unit. The *Mapper* may be used to identify correct targets for data units. When the *Mapper* was not used the first target in every unit is taken as a default target.

The data units may be provided as a *DataUnits* object or (especially in the case of REST interface) as a JSON document. The DataUnits class contains methods that allow to easily convert *DataUnits* object to JSON format and vice versa with the usage of GSON [14] library.

2.2. Filtering data

After all the training data has been loaded, it is possible to filter it using one of the predefined or a self-defined filter. The filter may just modify variable's values in all data units or may create a completely new dataset with less (or more) data units. The latter allows to reduce the amount of data e.g. to improve performance without losing accuracy.

2.3. Building a regression model

Building the regression model with usage of the training data is the core functionality of the ETCAL library. The model may be built by any implementation of the *Calibrator* interface. The responsibility of this object is to calculate model parameters using provided training data units and then return probable targets coordinates for a new data.

There are several calibrators already implemented including the 3rd degree polynomial one with an adjustable number of terms and the SVR calibrator using RBF kernel with parametrized *cost* and *gamma* parameters. There is also an abstract calibrator proposed that may be easily extended to use any regression function classifier implemented in WEKA [15] library.

2.4. Model evaluation

Evaluation may be done in three ways: (1) using training data, (2) using N-fold cross validation and (3) using some other testing data provided by user. The results of evaluation include: the determination coefficient (R^2), the root mean square error (*RMSE*) and the absolute error (all measured both in vertical and horizontal directions). The *Evaluator* object is implicitly used by optimizers typically using the cross validation (see the next section for details) — and may be also used directly when a user invokes the ETCAL's *checkErrors*() method.

2.5. Finding the best model for given data (optimization)

As it was mentioned above, there may be many different models built for the same input training data. Every model may be evaluated to check if it is better than other models.

For instance, for a model based on 3rd degree polynomial, the software is able to find the best set of terms that may be used for building the model. The problem is trivial for two input variables as there are only 10 terms and 1024 possible sets of terms, according to the combination with repetitions formula:

$$C_n^k = \binom{n+k-1}{k} \tag{1}$$

where n is a number of factors (4 in the case of 3 degree polynomial) and k is a number of input variables.

But when there are more input variables, checking all possible sets may be unfeasible. E.g. for 3 input variables there are over million combinations and for 4 input variables $3.44E+10$ combinations. Therefore, the software may use a heuristic algorithm to find the best model. So far only the genetic algorithm is implemented (using the JGAP library [16]) that effectively searches the space of all possible sets of terms. Similar technique is used for SVR models that for optimizing *cost* and *gamma* parameters.

As the decision which model is better is taken using only the training data it is very important not to over-fit the model. Therefore, the cross validation is a built-in option to the Evaluation object. It is possible to specify the number of folds and the type of cross validation.

2.6. Searching for correct target mappings

As it was already stated input data units may consist of more than one targets (possible gaze coordinates) for one specific set of variables V.

It happens when an implicit calibration is used instead of an explicit one [10]. In such the case a user is not forced to look at one specific point. Having information about the stimulation we try to *predict* where the user may look at.

The *Mapper* utilizes an assumption that the regression model built with correct mappings should give low errors for the training data. It creates calibrators for different mappings, compares errors and returns the mapping, which gave the lowest values.

When the number of possible mappings is too big, it is not possible to just check all mappings. Any heuristic algorithm may be used instead. The ETCAL library contains a build-in GeneticMapper that uses JGAP library [16] to find the best available solution, but users may create their own mappers by implementing an extremely simple *Mapper* interface.

2.7. Extendability

The users may easily create their own implementation of any of the aforementioned interfaces (*Filter*, *Mapper*, *Calibrator* or *Optimizer*). The new class may be applied in ETCAL in two ways. The first way is by creation of the object and invocation of an appropriate ETCAL method with this object as a parameter (e.g. *build*(*MyCustomCalibrator*)). The more flexible way is to invoke ETCAL method with a special generic container that encapsulates the class name and it is all parameters. For example to create *FourierFilter* class with two parameters *lowerBound* and *upperBound* the container would look like this:

```
{type: "org.example.FourierFilter",
params:{lowerBound:1, upperBound:14}}
```

It results in the execution of the following code:

```
Filter filter = new org.example.FourierFilter();
filter.setLowerBound(1);
filter.setUpperBound(14);
```

3. Illustrative examples

To evaluate usefulness of the ETCAL library there were several tests conducted as examples as to how the library may simplify the eye tracking data analyses. There were experiments provided for three different types of eye trackers: the self-made head mounted eye tracker using web camera (VOG), JAZZ-NOVO eye tracker and The Eye Tribe eye tracker. All tests has been explained in details in [17].

All eye trackers were used in points of regard calibration experiment. The participants of the experiment were looking at a stimulus presented on a screen. The stimulus was a circle pulsating on the screen to attract participant's attention. There were 30 subsequent stimulus locations (Fig. 3). The stimulus was displayed

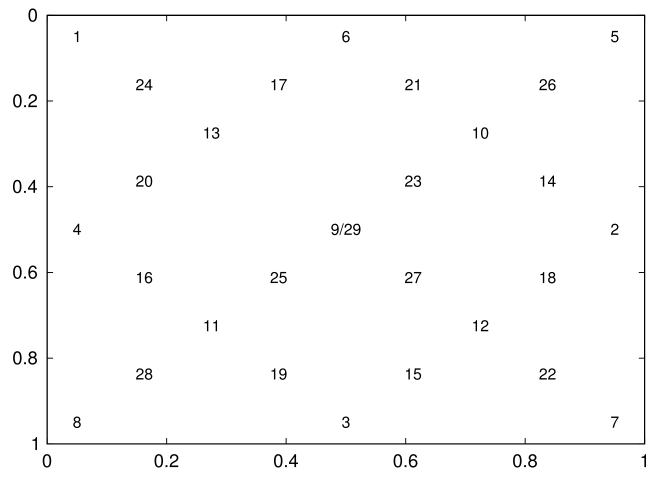


Fig. 3. Point's layout and order.

for about 3 s in each location. The order of stimuli presentation was the same for each session. For every eye tracker the same test was provided for different groups of participants.

Recordings obtained during each session were divided into two groups. First nine points locations were taken as a training set (9p) for a calibrator and the subsequent recordings of 21 point locations were taken as a test set (21p) to calculate errors. There was *CalibratorPolynomial* used to calibrate data with three different sets of terms. The terms for the calibrator when there are only two input variables are as follows:

$$\begin{aligned} x_g =& A_x x_e^3 + B_x y_e^3 + C_x x_e^2 y_e + D_x x_e y_e^2 \\ &+ E_x x_e y_e + F_x x_e^2 + G_x y_e^2 \\ &+ H_x y_e + I_x y_e + J_x \end{aligned} \qquad (2)$$

$$\begin{aligned} y_g =& A_y x_e^3 + B_y y_e^3 + C_y x_e^2 y_e + D_y x_e y_e^2 \\ &+ E_y x_e y_e + F_y x_e^2 + G_y y_e^2 \\ &+ H_y y_e + I_y y_e + J_y \end{aligned}$$

where gaze coordinates (x_g, y_g) are calculated based on two variables : x_e, y_e.

The cubic polynomial regression takes into account all terms (i.e. a mask is 1111111111), the quadratic polynomial only six terms (a mask: 0000111111) and the linear polynomial only three terms (mask: 0000000111). Fig. 4 presents plots and errors calculated for the cubic polynomial for both training (9p) and test (21p) data. When comparing it to results for the quadratic (Fig. 5) and linear (Fig. 6) polynomials it is visible that the polynomial with all 10 terms gave the best results for the training set while polynomials with 6 and 3 terms are not over-fitted and work better for unknown data (the test set).

The code that produces all six plots for the given session is quite straightforward:

```
DataUnits duTrain = DataUnits.load(<9p_session>);
DataUnits duTest = DataUnits.load(<21p_session>);
ETCal calsoft = new ETCal();
calsoft.add(duTrain);
ObjDef calibpar = new ObjDef();
calibpar.type = "pl.kasprowski.etcal.calibration.
CalibratorPolynomial";
// for three sets of terms
for(String mask:new String[] {"1111111111",
"0000111111","0000000111"}){
calibpar.params.put("mask", mask);
calsoft.build(calibpar);
//calibrate train
DataUnits duTrainCalibrated = calsoft.get(duTrain);
calsoft.plot(duTrain, duTrainCalibrated);
System.out.println("TRAIN:
"+calsoft.checkErrors(duTrain));
//calibrate test
DataUnits duTestCalibrated = calsoft.get(duTest);
calsoft.plot(duTest, duTestCalibrated);
System.out.println("TEST:
"+calsoft.checkErrors(duTest));
}
```

4. Impact

The decision to create and public ETCAL library was influenced by a fact that selection of a gaze determining model the best fitted to current data is still an unexplored problem. Although there are some papers that compare different solutions [18,12], there are no established, open, widely available and ready-to-use procedures to be applied to new data. There of course exist proprietary algorithms used by eye tracker vendors but they are useless for researchers that try to prepare their own calibration scenarios or their own eye trackers.

The paper presents the tool that is meant to help in choosing the best possible model for transforming data obtained from an eye tracker to gaze positions. Data subjected to such a processing may differ in several aspects: a registering device, a type of stimulus used, calibration scenario (both explicit and implicit) or a sampling frequency of a registration. Furthermore, the software may serve as a platform for the assessment of the chosen method efficiency. The main objectives to create ETCAL library were:

- Presenting a free and easy to use tool for people constructing their own eye trackers.
- Building a common platform to compare different calibration algorithms for given data.
- Establishing an open and strictly defined method for comparison of data quality.
- Giving opportunity to implement own elements: filters, calibrators, optimizers and mappers and to easily incorporate it into the ETCAL infrastructure.

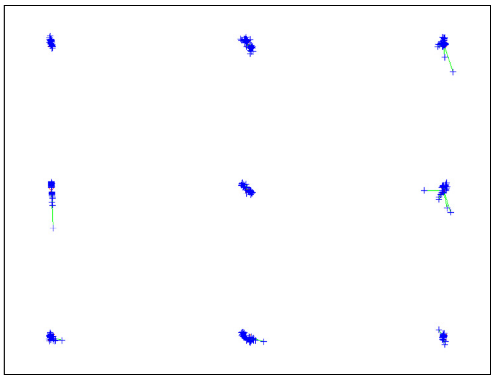
The important advantage of the software is that for the calibration model calculation it may be used out of the box and it is accessible via web interface that enables it to be used anywhere.

The main contribution of the software is as follows:

- Publication of a ready to use tool for the eye movement signal calibration with both programming and web interfaces.
- Publication of the heuristic algorithm useful for the implicit calibration when not all gaze positions (targets) are known.
- Presentation and implementation of the heuristic algorithm that tries to find the best model for a given data.

5. Conclusions

The ETCAL library presented in this paper and [17] is the ready-to-use tool that may be used to calibrate signals obtained from various types of eye trackers. It has provides built-in methods that are able not only to create a regression model but also to find the best set of parameters for a calibration function and even use a training data in which not all ground truth gaze coordinates are


(a) Training 9 points. $R_x^2 = 0.999$ $R_y^2 = 0.999$. $RMSE_x = 0.007$ $RMSE_y = 0.015$.

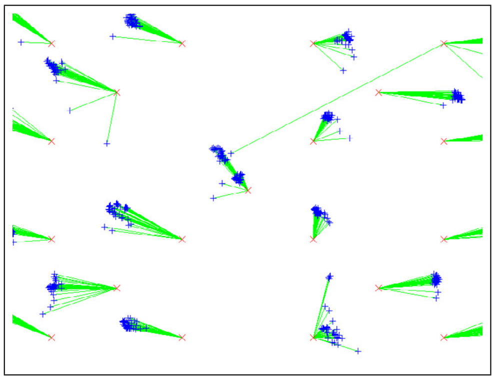
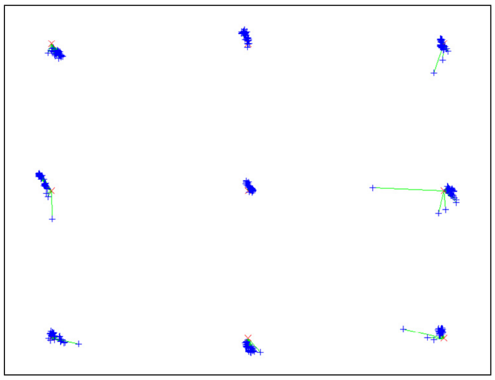

(b) Test 21 points. $R_x^2 = 0.859$ $R_y^2 = 0.963$, $RMSE_x = 0.090$ $RMSE_y = 0.046$.

Fig. 4. R^2 and *RMSE* errors for *X* and *Y* axes calculated for training 9p (a) and test 21p (b) sets after the calibration with the **cubic** (mask: 1111111111) polynomial calibrator. Red 'X' are predefined point's locations, blue '+' are points calculated by the calibration model. Green lines connect '+' with a corresponding 'X'.


(a) Training 9p. $R_x^2 = 0.998$ $R_y^2 = 0.995$. $RMSE_x = 0.017$ $RMSE_y = 0.025$

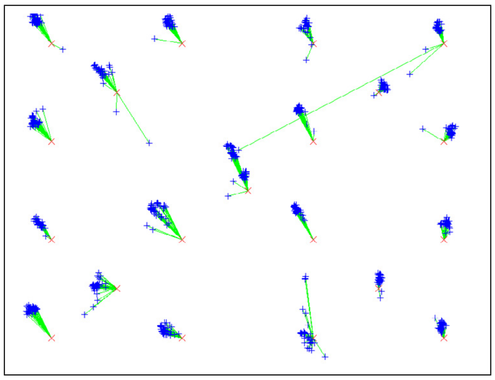
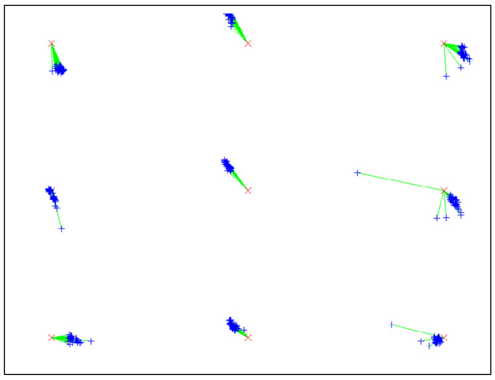

(b) Test 21p. $R_x^2 = 0.987$ $R_y^2 = 0.957$. $RMSE_x = 0.027$ $RMSE_y = 0.049$.

Fig. 5. R^2 and *RMSE* errors for *X* and *Y* axes calculated for training 9p (a) and test 21p (b) sets after the calibration with the **quadratic** (mask: 0000111111) polynomial calibrator. Red 'X' are predefined point's locations, blue '+' are points calculated by the calibration model. Green lines connect '+' with a corresponding 'X'.


(a) Training 9p. $R_x^2 = 0.990$ $R_y^2 = 0.980$. $RMSE_x = 0.037$ $RMSE_y = 0.052$.

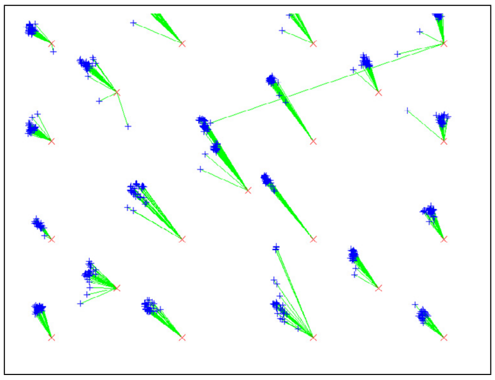

(b) Test 21p. $R_x^2 = 0.952$ $R_y^2 = 0.872$. $RMSE_x = 0.052$ $RMSE_y = 0.085$.

Fig. 6. R^2 and *RMSE* errors for *X* and *Y* axes calculated for training 9p (a) and test 21p (b) sets after the calibration with the **linear** (mask: 0000000111) polynomial calibrator. Red 'X' are predefined point's locations, blue '+' are points calculated by the calibration model. Green lines connect '+' with a corresponding 'X'.

known. It is fully extendable adding a new filter or calibrator is very easy and straightforward as it requires only the implementation of a simple interface. What is very convenient, the ETCAL may be used remotely through its REST interface, which makes it available for a software developed in any environment. It is also freely available to anyone interested from its github repository.

Due to all these advantages we hope that ETCAL may become a common platform for eye movement researchers to test and implement their own algorithms. We also have plans for the future development of the software by adding new improved algorithms for filtering, calibrating, optimizing and mapping an eye movement signal.

Acknowledgment

We acknowledge the support of Silesian University of Technology grant BK/263/RAu2/2016.

References

[1] Duque A, Vázquez C. Double attention bias for positive and negative emotional faces in clinical depression: Evidence from an eye-tracking study. J Behav Therapy Exp Psychiatry 2015;46:107–14.
[2] Armstrong T, Olatunji BO. Eye tracking of attention in the affective disorders: A meta-analytic review and synthesis. Clin Psychol Rev 2012;32(8):704–23.
[3] Najemnik J, Geisler WS. Optimal eye movement strategies in visual search. Nature 2005;434(7031):387–91.
[4] Bi Y, Shergadwala M, Reid T, Panchal JH. Understanding the utilization of information stimuli in design decision making using eye gaze data. In: ASME 2015 international design engineering technical conferences and computers and information in engineering conference. American Society of Mechanical Engineers; 2015 V02AT03A017–V02AT03A017.
[5] Kelly BS, Rainford LA, Darcy SP, Kavanagh EC, Toomey RJ. The development of expertise in radiology: in chest radiograph interpretation, expert search pattern may predate expert levels of diagnostic accuracy for pneumothorax identification. Radiology 2016;150409.
[6] Harezlak K, Rzeszutek J, Kasprowski P. The eye tracking methods in user interfaces assessment. In: Intelligent decision technologies. Springer; 2015. p. 325–35.
[7] Kasprowski P, Harezlak K, et al. Eye movement tracking as a new promising modality for human computer interaction. In: Carpathian control conference (ICCC), 2016 17th international. IEEE; 2016. p. 314–8.
[8] Vidal M, Bulling A, Gellersen H. Pursuits: spontaneous interaction with displays based on smooth pursuit eye movement and moving targets. In: Proceedings of the 2013 ACM international joint conference on pervasive and ubiquitous computing. ACM; 2013. p. 439–48.
[9] Celebi FM, Kim ES, Wang Q, Wall CA, Shic F. A smooth pursuit calibration technique. In: Proceedings of the symposium on eye tracking research and applications. ACM; 2014. p. 377–8.
[10] Kasprowski P, Harezlak K. Implicit calibration using predicted gaze targets. In: Proceedings of the ninth Biennial ACM symposium on eye tracking research & applications. ACM; 2016. p. 245–8.
[11] Zhang Y, Hornof AJ. Easy post-hoc spatial recalibration of eye tracking data. In: Proceedings of the symposium on eye tracking research and applications. ACM; 2014. p. 95–8.
[12] Kasprowski P, Harezlak K, Stasch M. Guidelines for the eye tracker calibration using points of regard. Inf Technol Biomed 2014;4:225–36.
[13] Harezlak K, Kasprowski P, Stasch M. Towards accurate eye tracker calibration–methods and procedures. Proc Comput Sci 2014;35:1073–81.
[14] GSON library, https://github.com/google/gson.
[15] WEKA library, http://www.cs.waikato.ac.nz/ml/weka/.
[16] JGAP library, http://jgap.sourceforge.net/.
[17] Kasprowski P, Harezlak K. ETCAL - a versatile and extendable library for eye tracker calibration. Digital Signal Process 2017.
[18] Blignaut P. Idiosyncratic feature-based gaze mapping. J Mov Res 2016;9(3).