A Cheap Portable Eye-Tracker Solution for Common Setups

Onur Ferhat and Fernando Vilariño
Computer Vision Center and Computer Science Dpt., Univ. Autònoma de Barcelona, Bellaterra, Barcelona, Spain

We analyze the feasibility of a cheap eye-tracker where the hardware consists of a single webcam and a Raspberry Pi device. Our aim is to discover the limits of such a system and to see whether it provides an acceptable performance. We base our work on the open source Opengazer (Zielinski, 2013) and we propose several improvements to create a robust, real-time system. After assessing the accuracy of our eye-tracker in elaborated experiments involving 18 subjects under 4 different system setups, we developed a simple game to see how it performs in practice and we also installed it on a Raspberry Pi to create a portable stand-alone eye-tracker which achieves **1.62**° horizontal accuracy with 3 fps refresh rate for a building cost of 70 Euros.

Keywords: Low cost, eye-tracker, software, webcam, Raspberry Pi

Introduction

Recent advancements in eye-tracking hardware research have resulted in an increased number of available models that have improved performance and that provide easier setup procedures. However, the main problem with these devices continues to be the scalability since their price and the required expertise for operation make them infeasible at the large scale.

These latest commercial models provide great accuracies (between 0.1 and 1°) at high frequencies (over 100Hz); however, in situations where such accuracies are not necessary and such frequencies are irrelevant, their high prices make them not a suitable choice. In this work, we aim to build a cheap, open source alternative that works on a hardware setup common in consumer environments: a standard webcam and an electronic device display. We believe that a system that provides comparable performance at an acceptable frequency will enable many applications on devices ranging from computers to tablets and smartphones.

We build our eye-tracker on top of the open source eye-tracker Opengazer (Zielinski, 2013), and our contributions are aimed at making the system more robust and increasing its performance. Our approach holds a good trade-off between robustness and real-time performance. Moreover, we provide the necessary tools to analyze the obtained results so that the assessment of accuracy will be easier for users.

This work was supported in part by the Spanish Gov. grants MICINN TIN2009-10435 and Consolider 2010 MIPRCV, and the UAB grants.

Method

The components of the software can be seen in Figure 1. The original system requires at least 4 facial feature points chosen manually on subject's face and it employs a combination of optical flow (OF) and 3D head pose based estimation for tracking them. The image region containing one of the eyes is extracted and used in calibration and testing. In calibration, the subject is asked to look at several target locations on the display while image samples are taken and for each target, an average eye image is calculated to be used as input to train a Gaussian process (GP) estimator. This estimator component maps the input images to display coordinates during testing.

Our first contribution is a programmatic point selection mechanism to automate this task. Then we propose several improvements in the tracking component. We finish the work on the blink detector to use these detections in other components. In calibration, we propose a procedure to assess and eliminate the training error. For the gaze estimation component, we try to employ a neural network method (Holland & Komogortsev, 2012). In the following subsections, we give the details of these contributions and talk about their effects on system performance in the discussion section.

Point Selection

Our contribution in the automation of the point selection mechanism aims at removing the errors due to operation mistakes. Moreover, it provides a standardized technique which increases the system's robustness. It employs a combination of Haar cascade detectors (Castrillón-Santana, 2012; Hameed, 2012), geometrical heuristics and a novel eye-corner detection technique. First, a cascade is used to detect the region containing both eyes and then the novel method detects the outer eye-corner points (Figure 2(a)). Here, the

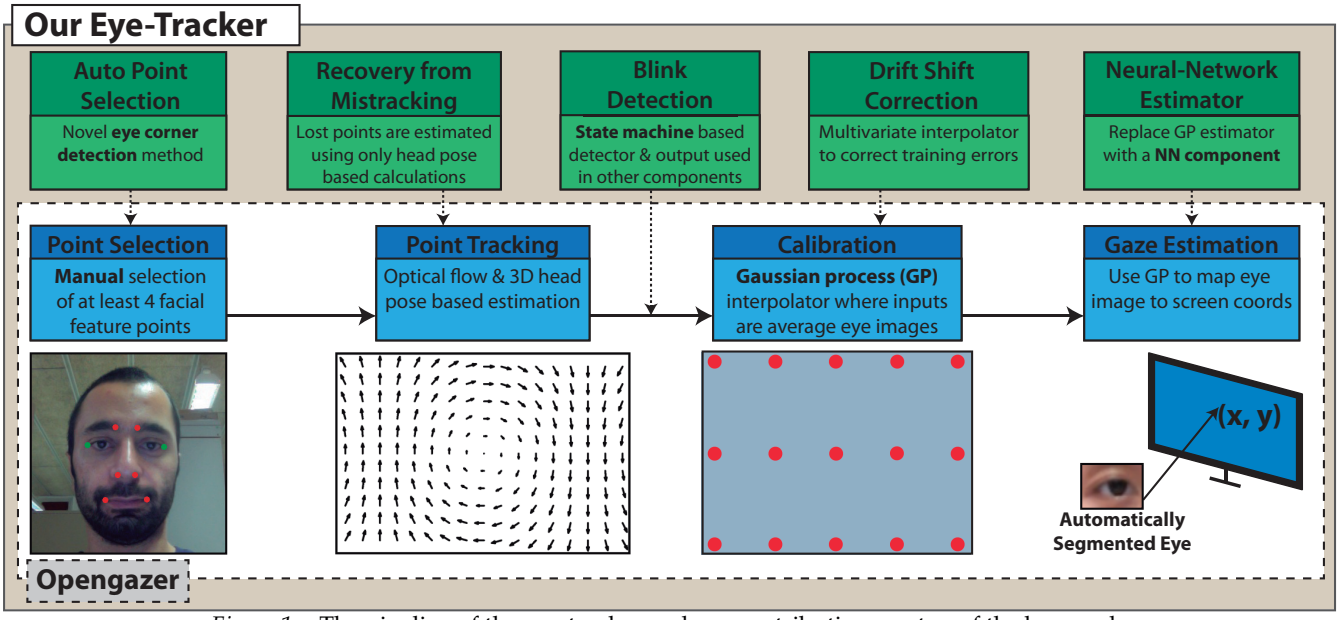


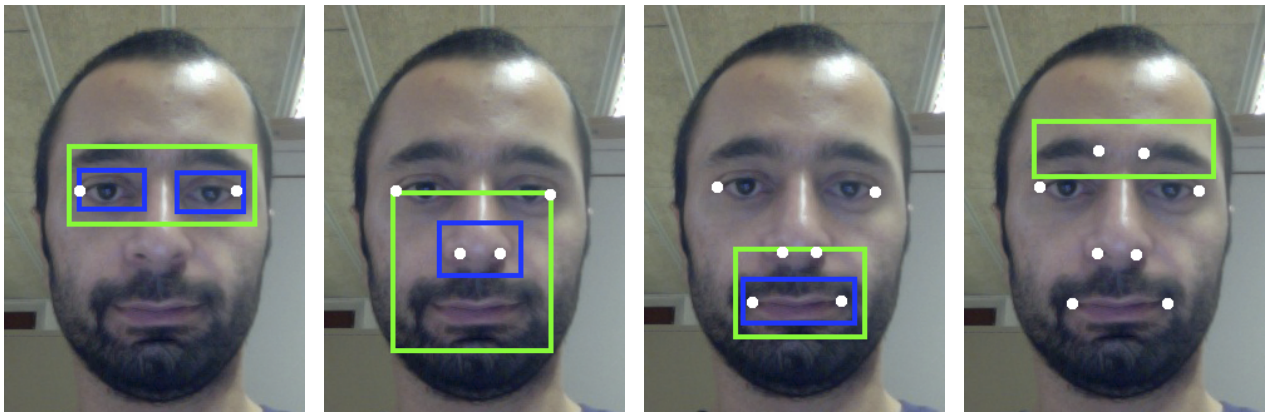
Figure 1. The pipeline of the eye-tracker and our contributions on top of the base code

proposed method extracts all corner points inside the ROI using Harris detector, and calculates the average corner coordinates in the left and right half of the region. These two points are considered as approximate eye centers and the outer corner points are chosen on the line that passes through them. As we only search a point around the eye corner that is stable enough, we do not make more complex calculations and we simply choose the eye corner points at a predefined distance ($1/3$ of the distance between two centers) away from the center point approximates.

After the eye corners are selected, we search the nose in a square region below them. When the Haar cascade returns a valid detection —as in the inner rectangle in Figure 2(b)—, the two nasal points are selected at fixed locations inside this area. The algorithm continues in a similar way for the mouth and eyebrow feature points.

Point Tracking

The point tracking component of the original system uses a combination of optical flow (OF) and 3D head pose based estimation. Optical flow calculations are done between the current camera image and the previous image. This methodology results in the accumulation of small tracking errors and causes the feature points to deviate vastly from their original positions after blinking, for instance. In order to make our eye-tracker more robust to these problems, we modified the tracking component so that OF is only calculated against the initial image saved while choosing the feature points. Moreover, if we still lose track of any point, we directly use the estimate calculated using the 3D head pose and correctly tracked points' locations.



(a) Eye corners (b) Nose tips (c) Mouth corners (d) Eyebrow corners

Figure 2. Sequence of facial feature point selection

Blink Detection

The blink detector is an unfinished component of Opengazer and we continue with analyzing it and making the necessary modifications to get it running. We believe that blinks have an effect on performance and by skipping them during training, we can remove the errors they introduce.

The blink detector is designed as a state machine with initial, blinking and double blinking states. The system switches between these, depending on the differences in eye images that are extracted as described in the previous section. These differences are calculated as the L2 norm between the eye images in consecutive frames. When the difference threshold for switching states is exceeded during several frames, the state is switched to the next state and a blink is detected.

We built on this structure and completed the rules for the state switching mechanism. Moreover, we added a state reset rule that resets the system to the initial state whenever the threshold criteria is not met at a certain frame.

Calibration

We propose a modification in the calibration part so that the images acquired during blinks are no longer included in the calibration procedure. This is crucial because these frames can alter the average eye images calculated during calibration and therefore are reflected as noise in the calibration procedure. However; as these frames are no longer available for calibration, we have to increase the time each target point is displayed on the screen in order to provide the system with enough samples during calibration.

Another improvement that we propose is the correction of calibration errors as illustrated in Figure 3:

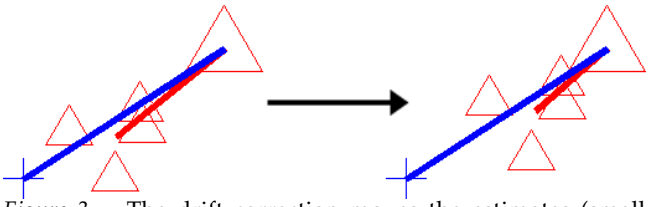


Figure 3. The drift correction moves the estimates (small signs) towards the actual target (larger signs). The training error direction (longer line) and testing error direction (shorter line) show the correlation.

Here, red triangles on the left side correspond to a target point displayed on the screen and the corresponding gaze estimations of our system, one for each camera frame. The larger symbol denotes the actual target, whereas the smaller ones are the estimates. The shorter line connects the average estimation and the target location. Therefore, the length and direction of this line gives us the magnitude and direction of average testing error. Apart from these symbols, the longer line that starts from the target denotes the direction of the calibration error. However, it should be noted that in order to easily observe the direction, the magnitude of the calibration error is increased by a factor of 5. In this figure, we can see the correlation between the calibration error and average testing error, therefore we propose a correction method. The final effect of this technique can be seen on the right side, where the estimates are moved closer to the actual target point.

To calculate the calibration errors, we store the grayscale images which are used to calculate the average eye images during calibration. Therefore, we save several images corresponding to different frames for each target point. After calibration is finished, the gaze estimations for these images are calculated to obtain the average gaze estimation for each target. The difference between these and the actual target locations gives the calibration error.

After the calibration errors are calculated, we continue with correcting these errors during testing. We employ two multivariate interpolators (Wang, Moin, & Iaccarino, 2010; *MIR*, 2012) which receive the average gaze estimations for each target point as inputs and are trained to output the actual target x and y coordinates they belong to. The parameters that we chose for the interpolators are: approximation space dimension = 2, Taylor order parameter = 6, polynomial exactness parameters = 1 and safety factor = 50. After the interpolator is trained, we use it during testing to remove the effects of calibration errors. We pass the currently calculated gaze estimate to the trained interpolators and use the x and y outputs as the corrected gaze point estimation.

Gaze Estimation

Originally, gaze estimates are calculated using the image of only one eye. We propose to use both of the extracted eye images to calculate two estimates. Then, we combine these estimations by averaging.

We also consider the case where the GP interpolator used here is completely substituted. Neural network (NN) methods constitute a popular alternative for this purpose. There exist recent implementations of this technique (Holland & Komogortsev, 2012). In the aforementioned work, an eye tracker using NNs to map the eye image to gaze point coordinates is implemented and is made available (Komogortsev, 2012).

We incorporated the NN method in our system by making use of the Fast Artificial Neural Network (FANN) library (Nissen, 2003) and created a similar network structure, and a similar input-output system as the original work. Our neural network had 2 levels where the first level contained 128 nodes (1 for each pixel of 16×8 eye image) and the second level contained 2 nodes (one each for x and y coordinates). We scaled the pixel intensities to the interval [0, 1] because of the chosen sigmoid activation function.

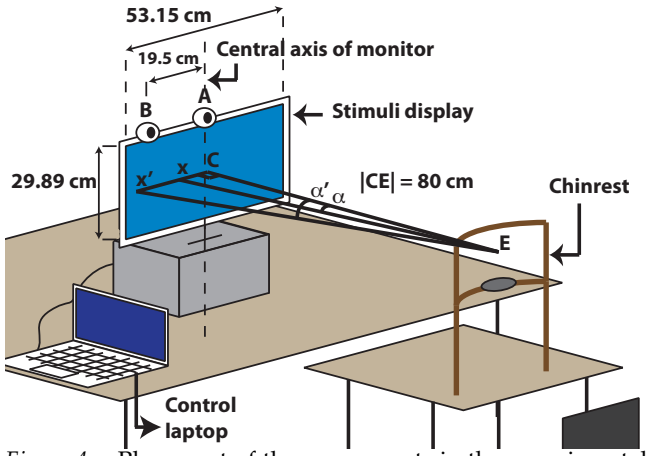
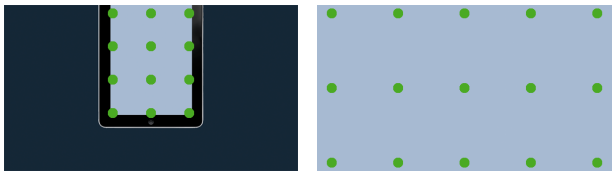


Figure 4. Placement of the components in the experimental setup and the geometry involved in error calculation



(a) iPad setup (b) Other setups

Figure 5. Target positions on the display for different setups

Other Contributions

Moreover, we propose a method to normalize the extracted eye images in order to correct the possible illumination variations in the environment. In this technique, we calculate the mean and standard deviation of pixel intensities in the extracted image and map the intensity distributions to a standardized distribution where pixel intensity mean is 127 and variance is 50.

Experimental Setup

In this section, we give the details of the experimental setup we created to test the performance of our application. Variations in the setup are introduced to create separate experiments which allow us to see how the system performs in different conditions. Figure 4 shows how the components of the experimental setup are placed in the environment.

The stimuli display faces the subject and it is raised by a support which enables the subject to face the center of the display directly. The camera is placed at the top of this display at the center (A), and it has an alternative location which is 19.5 cm towards the left from the central location (B). An optional chinrest is placed at the specific distance of 80 cm away from the display, acting as a stabilizing factor for one of the experiments.

By introducing variations in this placement, we achieve several setups for several experiments which test different aspects of the system. These setups are:

Standard setup: Only the optional chinrest is removed from the setup shown in Figure 4. Subject's face is 80 cm away from the display. The whole screen is used to display the 15 target points one by one.

Extreme camera placement setup: This setup is similar to the previous one. The only difference is that the camera is placed at its alternative location which is 19.5 cm shifted towards the left. The purpose of this setup is to test how the position of the camera affects the results.

Chinrest setup: A setup similar to the first one. The only difference is that the chinrest is employed. This experiment is aimed at testing the effects of head pose stability in the performance.

iPad setup: This setup is used to test the performance of our system simulating the layout of an iPad on the stimuli display. This background image contains an iPad image where the iPad screen corresponds to the active area in the experiment and is shown in a different color (see Figure 5(a)). The distance of the subject is decreased to 40 cm, in order to simulate the use-case of an actual iPad. The camera stays in the central position; and it is tilted down as seen necessary in order to center the subject's face in the camera image.

We also analyze the effect of different camera resolutions in these setups. This is done in an offline manner by resizing the original 1280×720 image to 640×480.

The error in degrees is calculated with the formula:

$$Err = abs(arctan(D_{xC}/D_{EC}) - arctan(D_{x'C}/D_{EC}))$$

where, x is the target, x' is the estimate, C is the display center and E is the face center point. The variables D_{xC}, D_{EC} and so on denote the distances between the specified points. They are converted from pixel values to cm using the dimensions and resolution of the display.

Results

In this section, we present the results which show the effects of the proposed changes on the performance. To achieve this, we reflect our changes on the original Opengazer code one by one and compare the results for all four experiments. We compare 6 different versions of the system which denote its certain phases:

1. **[ORIG]** Original Opengazer application + automatic point selection
2. **[2-EYE]** Previous case + average estimate of 2 eyes
3. **[TRACK]** Previous case + tracking changes
4. **[BLINK]** Previous case + excluding blinks during calibration
5. **[CORR]** Previous case + training error correction
6. **[NN]** Previous case + neural network estimator

Table 1
Errors in degrees for 1280×720 *camera resolution*

	Standard		Extreme		Chinrest		iPad	
Version	Hor. (σ)	Ver. (σ)	Hor. (σ)	Ver. (σ)	Hor. (σ)	Ver. (σ)	Hor. (σ)	Ver. (σ)
ORIG	**2.07** (1.69)	**1.73** (1.04)	**2.22** (1.68)	**2.07** (0.77)	**1.35** (0.63)	**1.61** (0.71)	**2.87** (1.66)	**2.49** (0.91)
2-EYE	1.94 (2.14)	1.67 (1.07)	1.70 (1.30)	1.82 (0.75)	1.18 (0.82)	1.46 (0.54)	2.54 (1.64)	2.46 (1.05)
TRACK	1.77 (1.52)	1.50 (0.80)	1.88 (1.15)	1.86 (0.66)	1.27 (0.85)	1.48 (0.56)	2.32 (1.29)	2.17 (0.75)
BLINK	1.78 (1.53)	1.49 (0.80)	1.89 (1.14)	1.88 (0.68)	1.27 (0.85)	1.47 (0.56)	2.31 (1.30)	2.14 (0.77)
CORR	**1.68** (1.56)	**1.43** (0.80)	**1.76** (1.15)	**1.78** (0.67)	**1.14** (0.79)	**1.36** (0.53)	**2.15** (1.32)	**2.02** (0.76)
NN	5.33 (2.83)	2.09 (0.89)	4.93 (2.48)	2.15 (0.46)	4.19 (1.03)	2.12 (0.64)	5.33 (2.83)	2.09 (0.89)

Table 2
Standard setup errors in degrees 640×480 *resolution*

	Standard	
Version	Hor. (σ)	Ver. (σ)
ORIG	**1.82** (1.40)	**1.56** (0.81)
2-EYE	1.56 (1.28)	1.48 (0.86)
TRACK	1.73 (1.37)	1.58 (0.78)
BLINK	1.73 (1.37)	1.58 (0.76)
CORR	**1.62** (1.40)	**1.50** (0.77)
NN	5.12 (2.26)	2.26 (0.92)

In all versions, the facial feature points are selected automatically by the method described in previous sections and gaze is not estimated during blinks. For each experiment, average horizontal and vertical errors for all subjects and all frames are given in degrees and the standard deviation is supplied in parentheses.

Table 1 shows the progressive results of our eye-tracker's performance for different versions of the system. Each result column denotes the horizontal or vertical errors for a different experimental setup. Moving from top to bottom in each column, the effects of our changes can be seen for a single error measure of an experimental setup. Along each row, the comparison of errors for different setups can be observed. Table 2 show the performance values of the system in the standard setup with the lower resolution camera. These results can be compared to the high resolution camera's results as seen in Table 1 to see how the camera resolution affects the errors in the standard setup. The original application's results (ORIG) and our final version's results (CORR) are shown in boldface to enable fast comparison.

Discussion

Considering the **1.68**° horizontal and **1.43**° vertical errors of the final system in the standard experimental setup, we conclude that we have improved the original system by **18%** horizontally and **17%** vertically. As seen in Table 2, the performance difference in the same experiment done with VGA cameras (**11%** horizontally, **4%** vertically) is comparably lower than the first case, which shows us that our contributions in this work exhibit more robust performance with the increased image quality. From another aspect, it means that better cameras will favor the methods we proposed in terms of robustness.

One interesting aspect of these results is that with the increased camera resolution, the original application shows a worse performance. We believe this is caused by the optical flow algorithm used in the tracking component. The increased detail in the images affect the tracking and the position of the tracked point may vary more compared to the lower resolution image. This, combined with the accumulated tracking error of the original application, result in a higher error rate. However, it is seen that the final version of our eye-tracker (**CORR**) recovers most of this error.

From the extreme camera placement setup results seen in Table 1, we see that shifting the camera from the top center of the display decreased the performance by **5%** horizontally and **24%** vertically. Here, the performance loss is mainly caused by the point tracking component. From such a camera angle, the farther eye corner point may be positioned on the face boundary, making it hard to detect and track. In order to tackle this problem, a 3D model based face tracking algorithm may be employed.

In the third experimental setup, we show that the use of a chinrest improves the performance by **32%** horizontally compared to the standard setup. Also the variance among subjects is reduced, which increases the reliability of this setup for experimental purposes. We also observed that in the vertical errors, this difference is not as significant.

The results for the iPad setup may be deceiving, because here actually the errors in pixels are lower; however, as the distance of the subject is used in the calculation of errors in degrees, the angular errors are higher. Each **1**° error in other setups corresponds to twice as many pixels on the screen compared to a **1**° error in the iPad setup. Using this rule of thumb, we can see that the iPad case results in lower error rate in pixels compared to even the chinrest setup.

We observe that excluding the blink frames from the calibration process (application version labeled BLINK) do not have a perceivable effect on the performance. We argue that the averaging step in the calibration procedure already takes care of the outlier images. The other component that failed our expectations is the neural network estimator, which performed up to 2.5 times worse than the Gaussian process estimator. We believe this is due to eye images extracted by our system. Currently the feature point selection and tracking mechanism allows small shifts in point locations and therefore the extracted eye images vary among samples. The GP estimator takes care of this problem during the image averaging step; however, the NN estimator may have problems when the images vary a little in the testing phase. In order to resolve this problem, a detection algorithm with a sub-pixel accuracy may be used to better estimate the eye locations.

Conclusion

Analyzing the changes in errors due to our contributions, we see that mostly we are increasing the performance. Our automatic point selection technique enabled us create an easy to use application, removing errors caused by wrong operation. The changes in the tracking component have proved to be useful in increasing the robustness. The experiments showed that the final system is more reliable in a variety of scenarios. The blink detection component is mostly aimed at preparing the eye-tracker to real world scenarios where the incorrect estimations during blinks should be separated from meaningful estimates. The proposed error correction algorithm helped the system better estimate gazes around the borders of the monitor. Apart from these experimental performance assessments, our work resulted in three additional valuable outputs:

A Portable Eye-Tracker: We installed our system on a Raspberry Pi as seen in Figure 6(a) and achieved a cheap (70 Euros) alternative to commercial eye-trackers. This system can be used as a separate input device to calculate and send the gaze point to the main computer. For experimental purposes, the necessary results analysis tools are also included in the device. The device has the same final error rates as in the experimental results (**1.62**° horizontally and **1.50**° vertically for 640×480 resolution) and has a **3 fps** update rate. By optimizing the code and using a faster camera, we expect to increase the refresh rate of the system.

An Eye-Tracking Video Dataset: The videos recorded during the experiments are gathered in a dataset of 12 subjects and 4 different experiment cases (subjects who have not given permission are excluded) (Ferhat & Vilariño, 2013).

The annotations for the videos denote on which frame the training and testing phase starts and the ground truth gaze points which are basically the positions of the target marker on the display at a given frame.

A Simple Game: We created a simple game where the users control a cursor by their gaze in order to find a frog in a cluttered environment. Once they fix their gaze on the frog's position, it is transported to another part of the display for the next episode. A demonstration of the game can be seen in Figure 6(b).

(a) Portable eye-tracker (b) The sample game

Figure 6. Other outputs of our work

References

Castrillón-Santana, M. (2012, September). *Modesto Castrillón-Santana.* Available from http://mozart.dis.ulpgc.es/Gias/modesto.html

Ferhat, O., & Vilariño, F. (2013, May). *CVC Eye-Tracking DB.* Available from http://mv.cvc.uab.es/projects/eye-tracker/cvceyetrackerdb

Hameed, S. (2012, September). *Haar cascade for eyes.* Available from http://www-personal.umich.edu/%7Eshameem

Holland, C., & Komogortsev, O. V. (2012). Eye tracking on unmodified common tablets: challenges and solutions. In C. H. Morimoto, H. O. Istance, S. N. Spencer, J. B. Mulligan, & P. Qvarfordt (Eds.), *ETRA* (p. 277-280). ACM.

Komogortsev, O. V. (2012, September). *Identifying fixations and saccades in eye tracking protocols.* Available from http://cs.txstate.edu/%7Eok11/nnet.html

MIR. (2012, September). Available from http://sourceforge.net/projects/mvinterp

Nissen, S. (2003). Implementation of a Fast Artificial Neural Network Library (FANN). *Report, Department of Computer Science University of Copenhagen (DIKU), 31.*

Wang, Q., Moin, P., & Iaccarino, G. (2010). A high order multivariate approximation scheme for scattered data sets. *Journal of Computational Physics, 229*(18), 6343 - 6361.

Zielinski, P. (2013, February). *Opengazer: open-source gaze tracker for ordinary webcams (software).* Available from http://www.inference.phy.cam.ac.uk/opengazer/