Cross-Platform Web Framework for Gaze Tracking

New opportunities for mobile interaction

Nicolai Harich[1], Simon Gebauer[1]
Computer Science and Media
Stuttgart Media University
Nobelstrasse 10, 70569 Stuttgart, Germany
contact@nicolaiharich.com, mail@simon-gebauer.de

Holger Schmidt, Gottfried Zimmermann
Responsive Media Experience Research Group
Stuttgart Media University
Nobelstrasse 10, 70569 Stuttgart, Germany
schmidtho@hdm-stuttgart.de, gzimmermann@acm.org

Abstract— **Gaze tracking functionalities provide a new way of interacting with electronic devices. As things are now, they are often limited to one specific device and require proprietary hardware and software. We propose a solution for web applications to counteract those limitations. By using the latest web technologies, we ensure platform independence. Due to the design and modularity of our solution, gaze tracking solutions become interchangeable. Web applications can easily consume gaze data provided by a generic event-based web-interface. The main contribution of this work is a universal calibration and mapping functionality. The raw data of miscellaneous gaze trackers can be processed and trans-formed, so that the gaze points are stabilized and correctly mapped onto screen coordinates. In the present paper, we describe the three-layer-architecture we use. Our work shows how to combine recent web technologies and algorithms to supply gaze tracking functionalities in a generic way.**

Keywords-Gaze tracking; cross-platform; calibration; eye tracking; gaze events; interaction; gaze-enhanced web; e-learning; real-time; adaptivity; adaptive e-learning

I. INTRODUCTION

In the last decades, eye tracking technologies were developed and established for analyzing human eye and gaze movements. In the fields of neuroscience, psychology and marketing, but also in the field of human computer interaction it became a widely accepted tool. However, the technology has not been accessible for the consumer market for a long time. No affordable solutions existed, since most systems were based on expensive hardware components. With ongoing development of algorithms and processing units, affordable software solutions reached the consumer market.

Even mobile devices are capable of gaze tracking now, but existing solutions are still very limited. The diversity of gaze tracking hardware and software makes it hard to integrate gaze tracking functionalities into existing applica-tions.

While trying to simplify the process of application development for desktop computers and mobile devices, a wide range of approaches emerged to handle this complexity. Applications can be written in different programming languages and even with web technologies to use them as “enriched web applications” in a browser.

[1] Authors contributed equally to the work

The latter one is advantageous for developing multi-platform applications, because a browser is available on almost every platform.

This motivates our research group to develop a gaze tracking framework based on web technologies. Previous works include a client-side web interface to connect mis-cellaneous gaze trackers to web applications. In this way, web applications can simply consume gaze events, without caring about the actual source.

Since gaze tracking affords high accuracy measurements and algorithms are quite sensible to various condition changes, most systems require regular re-calibrations. This makes it necessary to switch to a calibration software, which is usually supplied by the manufacturer of the eye tracker. In order to overcome this obstacle, we move the entire processing of the raw gaze data into the browser. The processing unit includes a calibration procedure and a mapping function, which transforms raw gaze data to display coordinates. This way the user can control the entire process through an arbitrary browser, without switching between applications.

The developed solution claims for general usage in web applications, but is actually part of the research project Adaptable and Adaptive Multimedia Systems (AAMS), which is an interdisciplinary research project driven by cluster 3 of the ScienceCampus Tuebingen [22]. The members of this cluster are from Tuebingen [23], Freiburg [24] [25] and Stuttgart [26]. The overall goal is to support self-regulated versus external-regulated learning from multimedia. Knowing that learning is a very individual process, future learning environments should automatically adapt to the needs of a user. In order to reach that target, advanced techniques have to be developed. Monitoring the gaze and eye movements allows to analyze the user’s learning behavior and momentary emotional state, which gives the opportunity to make adaptations on the system. Furthermore gaze trackers can act as input devices to enhance the accessibility.

Before describing our concept of in-browser calibration and mapping, we will give an overview of related work in the field of eye tracking technology in section II. In particular, we focus on the use of gaze tracking together with web applications like E-Learning environments. After that, we will introduce the frameworks we used or dev-eloped that enable web applications to consume gaze data provided by miscellaneous eye trackers as part of the systems architecture in section III followed by known issues in section IV.

Next we present the results of user studies we performed with two different gaze tracking systems in section V. The first system consists of a desktop computer and a high-cost, hardware-based eye tracker for professional use and the second system is an unmodified tablet with a front camera available for the consumer market. We evaluated the accuracy of both the stationary hardware system and the tablet with a user test to be able to compare those solutions.

We describe our E-Learning environment and motivate the usage of eye tracking in future applications like e-learning, accessibility or games in section IV.

We finally discuss the project in the conclusion in section VII.

II. RELATED WORK

Since eye trackers are entering the consumer market and even have been integrated in the first mobile devices, the development of gaze tracking applications got more and more attention. Besides traditional applications for analyzing the perception of users, a broad range of different applications appeared over the years.

The idea to use gaze tracking for interactive systems as an input device is actually quite old. One of the main challenges is to find a suitable way to allow the user to select or to confirm something. The first approach to solve the so called “midas problem” was dwell time selection, which was first suggested by Ware and Mikaelian [19]. The user has to fixate a target for a specific amount of time to perform a dwell time selection. Naturally this technique introduces some latency to the process. To ensure that the user is able to keep the fixation point inside the boundaries of the target until the dwell time is reached, the target has to be of a minimum size, which depends of course on the accuracy of the tracker.

Jacob [11] discussed different gaze interactions such as object selection, object movement and scrolling for text. Drewes [8] examined different gaze selection methods on mobile devices and introduced a new technique based on gaze gestures. Dybdal [9] investigated on the feasibility of different eye control techniques performed on small smartphone displays and compared them to alternative selection methods like finger-strokes and accelerometer-based techniques. Stellmach et al. [17] examined on different modalities of gaze-supported pan and zoom. A gaze-directed panning with mouse-wheel or touch-based zooming yielded high acceptance among the study participants.

There are several options to provide gaze data for web applications. Biedert et al. [4] developed a browser plugin to transfer the data from the eye tracker into the browser. Another browser-independent approach, developed by Wassermann, Hardt and Zimmermann [20], is a generic interface via HTML5 websockets. The interface technique of Wassermann et al. was developed as early part of the AAMS project and is also used in the scenario of using an SMI native eye tracker in combination with the newly developed in-browser calibration framework.

There are several promising ideas to use gaze data within web applications. Besides using the gaze data as an input device, it is possible to improve the usability of web applications by taking additional knowledge about the user’s interaction into account.

One idea is to use the user’s gaze point to make a prediction on what he will do next. Rozado’s, Shoghri’s and Jurdak’s studies [16] show that the gaze data can be used to predict which link is likely to be clicked next. With this additional information, the web content can be prefetched in order to speed up the browsing experience.

Alt et al. [2] investigated on adaptation and tailoring of web content based on gazing behavior. With a case study, they were able to prove that the adaptation induced a significant increase of the user’s attention.

Similar attempts exist in the field of E-Learning environments. Several research groups investigate on the integration of eye tracking into E-Learning environments [7] [18] [15]. Copeland and Gedeon [6] investigated on the gaze-based prediction of reading comprehension. Their results have shown, that the number of fixations and total duration of fixations are suitable measures for subject familiarity and extent of answer-seeking.

E-Learning environments can make use of such measurements to offer additional assistance to the learner whenever suitable. One example is iDict, a translation aid for language courses, which displays tooltips to support the learner in case difficulties arise [10].

III. ARCHITECTURE

Gaze trackers can be classified in two main categories. Software-driven solutions have the advantage of being independent of additional devices except for a camera, thus reducing expenses. In contrast, most external modules include a dedicated capturing engine like an infrared camera with a special illumination unit, which makes the recognition of pupils more robust.

During the development of the framework, we used two different devices. On the one hand a hardware eye tracker from SensoMotoric Instruments (SMI), on the other hand an Amazon Kindle Fire tablet with a built-in front camera.

To be able to use both (or even other) devices for gaze tracking, we developed a three-layer-architecture: layer 1, which includes the gaze tracking hardware with its drivers, layer 2 including different ways to enable the communication between layer 1 and 3, and layer 3, the JavaScript-based gaze data processing unit as a web frontend which receives the gaze data from the communicator and provides functionalities for third party applications to use the gaze data.

Using this three-layer-architecture, gaze tracking functionalities can be provided on different devices and for different applications. These three layers are exchangeable if required, e.g. irrespective of the underlying hardware (layer 1), the communicator can be enabled to send the gaze data to the frontend. Moreover, the application that uses the gaze data can be exchanged, e.g. it does not matter whether it is an E-Learning-platform or a game. Fig. 1 shows the three-layer-architecture.

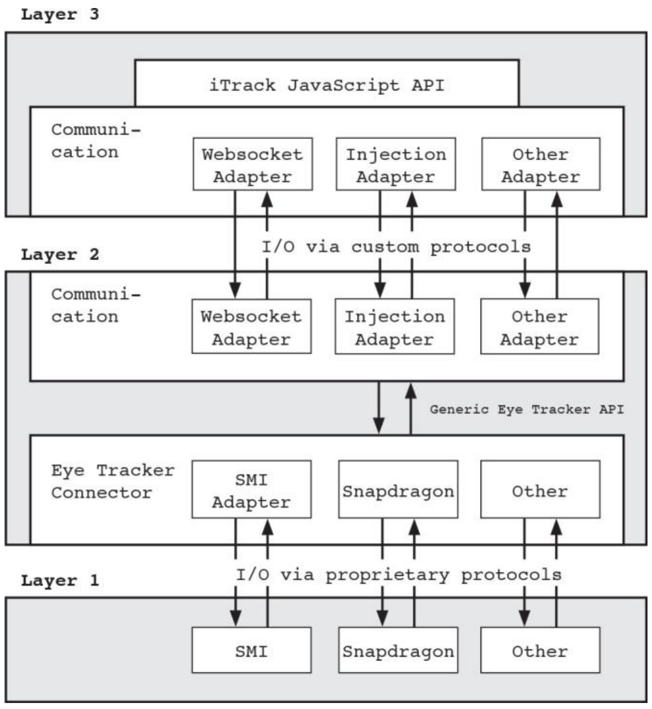

Figure 1. The three layer architecture guarantees interchangeability.

A. *Layer 1: Hardware Layer*

The base layer is the gaze tracking hardware with its proprietary drivers. Whether you use a purpose-built gaze tracking hardware, a desktop computer with a webcam or a mobile device with a built-in camera, you still need drivers to receive image data from the camera.

The SMI eye tracker is a typical, purpose-built gaze tracking solution which consists of the gaze tracking hardware and the proprietary software. The SMI eye tracker works with a video camera in combination with an infrared LED, which creates a reflection on the user's eyeball. The vector between the reflection and the centre of the pupil can be used to calculate the direction of the gaze. We use the iTrackServer as described in the following section to capture the gaze data in order to use it in the context of the three-layer-architecture.

The tablet is equipped with a front camera, which can provide a video stream of the user. By detecting the eyes in the image, it is possible to compute the gaze point of the user. We use the Snapdragon SDK [1] to do this in an efficient way. The SDK includes several facial processing algorithms which can be executed in real time. The Snapdragon SDK is compatible with Android devices equipped with appropriate Snapdragon processors.

For the gaze tracking functionality, we use the face recognition module of the Snapdragon SDK. Properties of detected faces are stored in FaceData objects, which includes the following information:

- Blink Detection - information about how wide the user's eyes are opened
- Gaze - information about the gaze point of the user
- Smile Value - probability of the user's smile
- Face Orientation - orientation of the user's head in all three axes

For our purpose we are mainly interested in the provided gaze points. They are presented in a normalized, two-dimensional camera space with values between -1 and 1. The origin is defined by the optical axis of the camera. We implemented a mapping algorithm to map this raw data onto screen coordinates, seen in Fig. 2, which is described below.

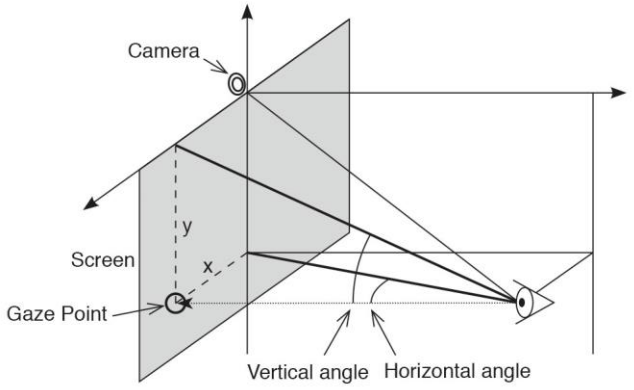

Figure 2. Visualization of gaze tracking

Since we want to use the gaze data in web applications, we embedded a webview [3] in our native application. The data is then delivered in a JSON format by injecting code into the webview. We will describe this in more detail in the next section.

B. *Layer 2: Communication Layer*

Layer 2 is a communication interface between layer 1 and layer 3, for sending the uncalibrated raw gaze data from the gaze tracker to the gaze-data processing unit. This interface is implemented in a generic and customizable way, ensuring that hardware and processing software can be exchanged.

Benjamin Wassermann and Adrian Hardt from the Stuttgart Media University developed a universal interface called iTrack [20] as part of the project AAMS. It was mainly implemented to provide an interface between data from SMI eye trackers and a browser. It consists of two parts: an iTrackServer and a JavaScript file called itrack.js. The iTrackServer is a command line tool, providing a websocket server to forward the data from the SMI software to a browser. The corresponding JavaScript is embedded into a web page to receive data from the websocket server.

Websockets offer a bidirectional communication between different hardware and software, but Android prohibits a device-local websocket communication between an Android web view and its wrapping native application for security reasons. This fact denies the use of a native webbrowser while having access to the systems front camera. In contrast, a webview does not have these limitations.

Moreover, building a client-server-architecture on a single device to establish a communication between native Android code and the contained webview is not an appropriate solution. This is why we developed a second way to let layer 1 communicate with layer 3: an injection adapter based communicator (injection communicator).

To establish a reliable connection between Android code and the contained webview, we extended the iTrack library with an injection communicator which uses JavaScript event emitting and event handlers. Depending on the availability of websockets, layer 2 provides a fallback mechanism in the form of the injection communicator, enabling the layer 1 software to send the gaze data by triggering JavaScript events.

According to the availability of the communication interface, layer 3 automatically chooses the data source to handle the raw gaze data.

C. Layer 3: JavaScript-based processing

Irrespective of the underlying gaze tracking hardware or communication, layer 3 is implemented in JavaScript to work in a browser. This browser can be a typical browser on a desktop computer or a webview integrated in a mobile App. Due to the usage of the extended iTrack library, layer 3 is able to automatically choose the data source through its fallback mechanism. If neither the websocket, nor the injection communication is available, layer 3 can simulate incoming data by using the mouse or touch coordinates as gaze data for testing purposes.

Like JavaScript emits mouseIn or mouseOut events when the cursor enters or leaves an object in the browser, the iTrack library emits gaze events like gazeIn or gazeOut. These events can be handled by conventional event handlers in JavaScript, providing a gaze-tracking API that is usable in different contexts.

Since the iTrack library requires previously calibrated and mapped gaze data (as provided when using a calibrated eye tracker), we extended the library to be able to handle raw gaze data.

The incoming gaze messages contain two-dimensional coordinates of a gaze point. Depending on the hardware and its drivers, this data might be normalized within distinct limits (e.g. [-1;1]), having its origin defined by the optical axis of the camera (Figure 2), but it is not implicitly the case. Therefore the calibration procedure has to be able to handle different scales and offsets of the raw data. Next we will describe our calibration procedure for mapping the (raw) gaze data to screen coordinates.

1) Calibration

Mapping the gaze data to screen coordinates can be done in different ways. Hardware suppliers like SMI provide their own software to calibrate and map the raw data, depending on particular hardware.

To be independent from proprietary software and to meet the requirements of different display sizes, pixel densities and camera positions, we developed a semi-automatic calibration procedure which runs in a webbrowser or webview. In the following, we give an overview of our calibration procedure.

While calibrating, the user is asked to gaze at reference points shown on the display, see **Figure 3**. These points are successively shown for a short while to collect the user's gaze data. The recording of gaze-data is shortly interrupted after the reference position moved to avoid wrong measurements due to the latency of the user's focus.

The resulting gaze data is usually noisy and contains some outliers. A simple solution to identify outliers is to look at the distances between each acquired point and the centroid of all points.

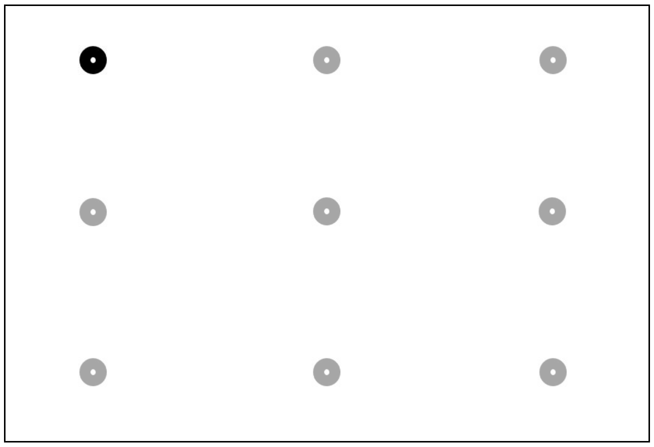
Figure 3. Screenshot of our calibration screen. We subsequently display a target point (grey targets are in fact currently invisible).

Those distances are compared to the mean of all distances among points. Points with a large relative distance are rejected from the further processing. The known screen coordinates of the reference points and the collected raw gaze data are used to approximate the transformation parameters for both horizontal and vertical direction. This is a straightforward fitting problem, called linear regression, which can be solved by a least-squares technique [27].

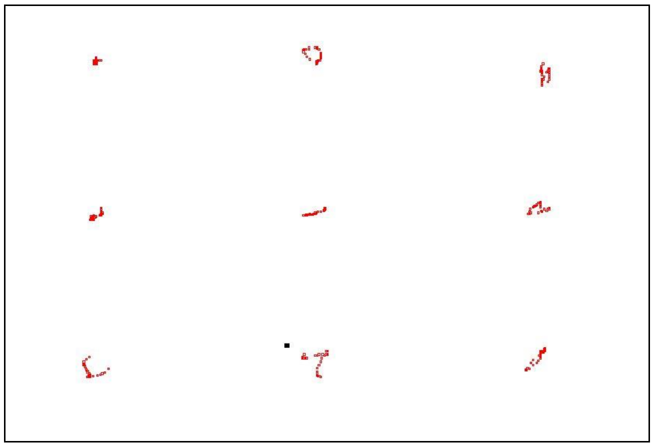
Figure 4. Acquired (noisy) data points, which build the basis to fit the mapping functions.

Since the accuracy of the calibration is influenced by conditions of attendance, such as head movement or correct gazing, a stability index is computed to check the quality of the calibration. If the index falls below a certain threshold, the user is informed about the problem and the calibration process will automatically restart.

Our described approach is quite simple and could be replaced by a more advanced technique. For example, a support vector machine (an advanced learning algorithm for classification and regression analysis) could be used to learn a mapping function from the data. During the development process we tested a so called epsilon-SVR to fit a mapping function. The epsilon-SVR has the advantage that it is more robust against outliers and therefore does not need a pre-filtering step. For our testing purpose, we used the Java implementation of libsvm [5] on the tablet. The improvements were relatively low in comparison to the loss of performance. Therefore, we kept our first approach.

2) *Mapping and Filtering*

The x- and y-components of the raw gaze points are mapped according to two independent, linear functions. The parameterization is provided by the previously described calibration procedure.

The signal of a gaze tracker can be relatively unstable, because it includes a certain amount of noise and also measures micro saccades of the eye. Especially, when we use the data as a pointing device, we want to smooth those small changes, as they are very disturbing for the user. Conventional smoothing algorithms, like moving averages (a series of averages of different data subsets), are unsuitable for this task, since those introduce large delays subsequent to a saccade. For this reason, we have to treat fixations and saccades separately. Furthermore, we have to ensure robustness regarding outliers. Špakov [14] compared several eye movement filters for the usage in HCI applications. The comparison was based on the introduced delay, smoothness and closeness to the idealized data. The outcome of this study was that algorithms with state detection (“fixation” and “saccade”) and adapted processing generally performed better than others. The type of smoothing we use during a fixation is less important and can be quite simple. Kumar presented an algorithm [12], which uses a one-sided triangular filter to smooth the data points during fixations. For each new data point, we determine whether it is the start of a saccade, a continuation of the current fixation or a single outlier during a fixation. For further readings and instructions, we refer to the existing works [12] [13].

In addition to the smoothing effect, the algorithm indirectly generates information about the start and end of fixations. This side product can be used to generate high level gaze events as proposed by Wassermann et al. [20].

In addition, one could consider more specialized filtering solutions. For example, the stability could be enhanced for gaze pointing tasks, such as activating a button on a website, by including knowledge about the position of such gaze sensitive areas in the algorithms. Reasonable results were obtained by a “speed reduction” algorithm such as described by Xinyong, Xiangshi and Hongbin [21]. This method reduces the risk that the cursor temporarily leaves the target area due to eye jitter or noise. Especially dwell time selection benefits from this behaviour. Since our solution should be as universal as possible, we did not directly include the algorithm in our framework.

IV. Known issues

Gaze tracking is complex, especially on mobile devices without purpose-built hardware. A reasonable use in applications demands in general high reliability and robustness. There are additional requirements depending on the purpose. Real time applications require a high-frequent data supply and applications such as gaze pointing require high precision.

A. *Performance*

Since the framework is based on JavaScript, its performance depends on the JavaScript performance of the device. As things are now, the performance of the Android webview [3] we used on the Kindle Fire tablet is limited: our measurements have shown, that the webview is able to process around four frames per second, irrespective of the application which uses the gaze data.

B. *Latency*

The latency between a user’s gaze and the mapped result plays an important role. When using gaze tracking as a real-time application, the latency must be as low as possible. We developed our gaze tracking framework and the communication layer with low-latency in mind: single gaze messages can be dropped if the frontend is not able to process those. Since the performance of the webview is limited, the latency can grow up to about 250 ms, which makes real-time applications unusable.

Latency is also caused by smoothing algorithms, especially when large time windows are used. Therefore filtering algorithms should be parameterized, such that a reasonable balance between latency and smoothing is given. In addition, it is important that algorithms treat saccades and fixations differently. It is evident that fixations require stability, whereas a saccade demands an immediate response.

C. *Accuracy*

The accuracy of a gaze tracking solution means that the distance between the user’s actual gaze point and the calculated gaze point on the screen should be as small as possible. It is influenced by many factors, which can be classified into algorithmically minimizable influences, unavoidable hardware influences and outside influences.

Lens-distortions for example can be minimized by algorithms, whereas the display size or the monocular camera position cannot be manipulated. Outside influences, like the user’s position, his distance to the screen and movement or influences like light conditions, can be optimized but not avoided.

The degree of influence of these factors vary, depending on the eye tracking hard- and software.

V. Testing with users

As described before, one of the most important parts of a gaze tracking solution is its accuracy. Within this project, we developed a gaze tracking software for a mobile device, extended the iTrack library and built an in-browser calibration algorithm. We developed an appropriate browser-based calibration checker software to be able to give a general statement about the accuracy of this solution.

Due to the fact that the framework works on both, desktop computers and mobile devices, we are now able to compare (a) the desktop solution with a hardware-based eye tracker, (b) the hardware-based eye tracker combined with the in-browser calibration, (c) the mobile solution with the software-based eye tracker and a calibration process written in native Android code and (d) the software-based eye tracker combined with the in-browser calibration.

TABLE 1. OVERVIEW OF OUR DIFFERENT TEST-SETTINGS

calibration	Desktop	Mobile
native	a	c
in-browser	b	d

We invited people to an appropriate user test to find out if the results are comparable and probably determine a factor to compare the accuracy of those solutions. We tested the four different calibration methods with 45 subjects. Most of them had never used an eye tracker before, neither a stationary one nor a mobile one.

A. *Testing a calibration*

It is at first required to calibrate the gaze tracking software to test a calibration. The calibration shows reference points on the display and meanwhile collects gaze data of the subject to determine the mapping function from raw data to gaze points. The calibration checker acts in a similar way, after collecting data it saves the reference points' coordinates with the actual gaze points' coordinates in a plain text format. Subsequent to the user test, we aggregated and evaluated the collected data.

B. *Test conditions*

Both, the stationary eye tracker and the mobile device, were placed on a table to avoid influences of movement of the hardware. The subjects sat on chairs and were told to move as little as possible. We tested in a low-distraction environment and under constant conditions concerning lighting to reduce ambient impacts on the results.

C. *Test procedure*

Every subject was asked to test both, the stationary gaze tracker as well as the mobile solution. Both devices support a native calibration and our in-browser calibration, so every subject had to calibrate and evaluate four times. We shuffled the order of calibration processes randomly between subjects to avoid systematic errors.

In every situation, the calibration procedure was started first. Afterwards, the calibration checker was started to collect data for the evaluation of the calibration.

SMI provides a proprietary calibration method built into the software shipped with the hardware. Since SMI does not disclose its calibration algorithms, we cannot provide any information on them.

When calibrating the SMI device, the software creates a plain text configuration file. This file contains calibration parameters in the form of (a) screen points from the calibration, (b) numeric RAWLEFT and RAWRIGHT values and (c) numeric COEFFLEFT and COEFFRIGHT values and is used by the SMI software to map the raw data to screen coordinates. The iTrackServer then sends the resulting data of the software to a websocket, which makes it impossible to use the SMI eye tracker raw data in an uncalibrated state. Since the in-browser calibration needs to be tested with an uncalibrated gaze tracker, we overwrote the values of (b) and (c) in the mentioned file with random numbers. The resulting calibration file was then loaded into the SMI calibration software to simulate an uncalibrated state.

D. *Evaluation*

The collected data consists of plain-text files containing the two-dimensional coordinates of the reference points and the actual gaze coordinates for each of the calibration methods. The absolute distance between the reference point and the actual gaze point describes the mean absolute error (MAE).

TABLE 2. STATISTICAL DATA OF THE EVALUATION

	Desktop		Kindle	
	Browser	Native	Browser	Native
MAE [px]	89.6	80.0	176.9	163.7
Std deviation [px]	15.7	14.9	27.2	25.0
Sample size	20,812	20,330	19,260	18,780
Outlier count	271	977	1,023	834
Outlier-ratio [%]	1.30	4.81	5.31	4.44

The aggregated data contains between 1.30 % and 5.31 % outliers, which are points with distances greater than the threshold of 500 pixels (marked grey in TABLE 3 below). This threshold is about half the display size of the mobile device, so we decided to reject those outliers from further processing. These outliers can occur (a) through distraction of the subject, followed by a gaze point outside of the calibrated area or even outside of the monitor, (b) by saccades of the eyes of the subject, which are not measureable or (c) due to measuring inaccuracy of the gaze tracking hard- or software (also due to external influences like reflections).

TABLE 3. NUMBER OF GAZE POINTS WITH A DISTANCE TO THE REFERENCE GREATER THAN GIVEN ERROR

	Desktop		Kindle	
Error [px]	Browser	Native	Browser	Native
> 0	9,196	8,712	8,160	8,160
> 1	9,196	8,711	8,160	8,160
> 10	9,195	8,532	8,096	8,136
> 100	5,749	1,852	5,351	5,610
> 500	170	401	414	475
> 1,000	63	231	155	82
> 10,000	3	205	0	0
> 100,000	3	205	0	0

Excluding those outliers, the mean distances between the reference points and the gaze points vary from about 80 pixels (desktop) up to about 180 pixels (mobile device). In Figure 5, these mean absolute errors are shown with the standard deviation (SD). Compared to the desktop eye tracking solution, the mean distance between the reference and the actual gaze point of the mobile solution is more than twice as high which gives hints to a greater inaccuracy of the webcam based gaze detection quality.

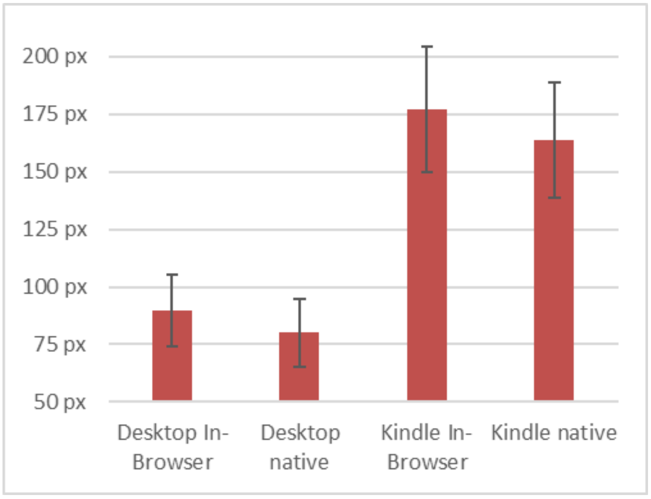

Figure 5. Mean absolute error (MAE) between the reference points and the gaze points in pixels with standard deviations

Figure 6 shows the calibration result from one of the test persons with the SMI eye tracker using the in-browser calibration. Crosses illustrates the reference points shown on the screen, the point near each of the crosses is the collected gaze data. Figure 7 shows the data from the same subject using the mobile gaze tracker with the in-browser calibration as well. Again, the cross illustrate the reference points, whereas the collected gaze data is shown as squares (upper reference), triangles (right), lines (lower) and circles (left).

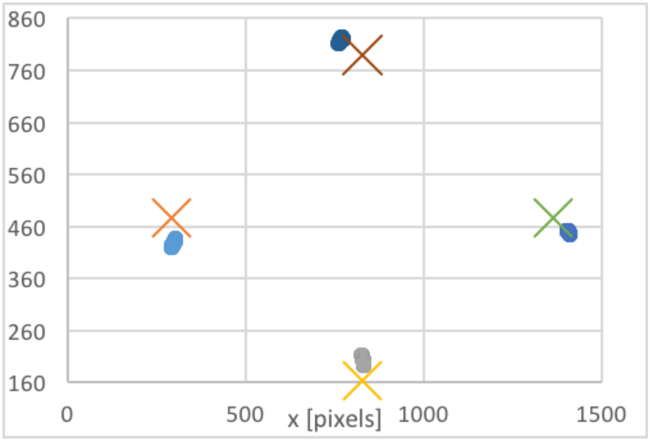

Figure 6. Calibration checker result from the SMI eye tracker, the cross show the target, the points show the actual values

As can be seen in the graph, the accuracy of the mobile gaze tracker using the same algorithm as the desktop is comparatively poor. About 26 % of the collected gaze points show an error less than 100 pixels, which is approximately one third of the amount of the desktop's native eye tracker (In-Browser calibration: 64 %, native calibration: 74 %).

The desktop eye tracker is a solution for professional use, providing binocular optics with purpose-built infrared illumination and proprietary software. In contrast, the mobile solution is built from a consumer tablet with a single front camera. Keeping these facts in mind, the mobile solution is in fact not as accurate as the desktop solution, but quite usable to get an estimate of the gaze point of the user. In the next section, we introduce possible applications for eye tracking.

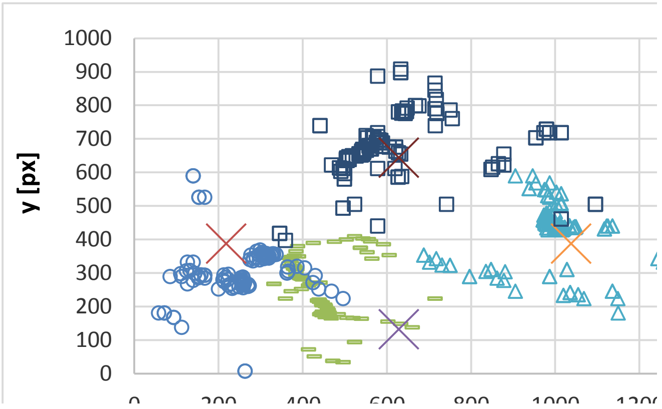

Figure 7. Calibration checker result from the Kindle tablet, the cross show the target, the other signs show the actual values

VI. Possible Applications

Eye tracking does not necessarily require high precision gaze tracking hardware. Depending on the context of use, it might be sufficient to be able to distinguish a few screen points.

A. *E-Learning*

In the area of E-Learning, gaze tracking provides new possibilities for both, the learners and lecturers. For instance, the individual learning process could be supported by automatic adaption of content or interfaces. By deriving information about the emotional and cognitive state of a user, we can adapt the learning environment and content to their specific needs. For example, the system can provide additional information (e.g. a "Bubble Help") if it discovers that the user has problems in understanding the content. It is also possible to classify different types of learners (e.g. "visualizer", "verbalizer", "intermediate") in order to be able to present the knowledge in the most suitable form. Furthermore the accessibility can be extended by supplying an additional input device.

The lecturer can also take advantage by better understanding how people learn. He can evaluate peoples' eye movement with heat maps, determining which parts of the learning matter seem to be the most interesting parts for a particular learner.

B. *Accessibility*

Moreover, gaze tracking is an interesting field of research for accessibility. Providing hardware-independent gaze tracking can optimize functionalities on websites or in software. People with disabilities can take advantage of this technology, even if they do not own purpose-built gaze trackers.

Providing different ways to navigate through web contents improves the accessibility of a website. However, it is not standardized how to connect miscellaneous gaze trackers with web applications, which can be done using our gaze tracking framework.

C. Games

Games, especially browser-based games, can take advantage from this technology to provide a new user experience and enable attention-awareness. Using eye tracking as an input device, it is possible to replace the mouse or keyboard input with gaze data, e.g. for moving around in a virtual world.

Furthermore, gaze-supporting games can respond to the user's gaze with changing graphics, e.g. supply additional information for the gameplay when looking in one of the screen corners. An interdisciplinary field of use are serious games and e-learning games.

VII. Conclusion and Outlook

In this work we have presented a concept of providing gaze data to web applications in a generic way. The concrete implementation of the calibration functionality is kept intentionally very simple and could be substituted by more advanced algorithms. It is also thinkable, that other gaze trackers afford higher polynomial mapping functions to achieve accurate results.

User tests proved the capability of our approach for two possible systems. We looked at two extremities, a desktop computer in combination with a purpose-built eye tracker and a low-cost solution on an unmodified tablet without any external hardware.

As expected, the accuracy is highly dependent on the quality of the supplied raw data. Inaccuracies actually limit the practical use in applications. Primarily real-time usage, such as forms of user interactions, are very critical and demand high accuracy and low latency.

For the purpose of antagonizing this issue, one could deploy alternative user-interaction techniques. In some case it may be feasible to use eye or head gestures. Gestures do not require absolute precision and are therefore more robust with regard to noise and calibration inaccuracies. In spite of those advantages, gestures are not always suitable and cannot fully substitute a gaze pointer. On the other hand, we expect that the accuracy, especially of low-cost solutions, will be improved dramatically in the near future.

With this work, we made a contribution for the AAMS project towards innovative, auto-adaptive E-Learning environments of tomorrow. The very next steps will include the integration of our framework into the E-Learning system developed and used in the project AAMS called Adaptive Learning Module (ALM) and the implementation of several gaze tracking-based functionalities.

Acknowledgments

This research has been conducted within the ScienceCampus Tübingen "Informational Environments", an interdisciplinary research collaboration of the Knowledge Media Research Center and the University of Tübingen, with funding from the German state of Baden-Württemberg, Ministry of Science, Research and the Arts. This publication reflects only the authors' views and the funding agency is not liable for any use that may be made of the information contained herein.

We thank all students, who voluntarily participated in our user studies.

References

[1] Qualcomm Technologies Inc. *Snapdragon SDK for Android*. [Online] Available from: https://developer. qualcomm.com/software/snapdragon-sdk-android.

[2] F. Alt, A. Shirazi, A. Schmidt, and J. Mennenöh. "Increasing the user's attention on the web". In *the 7th Nordic Conference*, 544. DOI=10.1145/2399016.2399099.

[3] Android. *WebView*. [Online] Available from: http:// developer.android.com/reference/android/webkit/WebView. html.

[4] R. Biedert, G. Buscher, S. Schwarz, M. Möller, A. Dengel, and T. Lottermann. "The text 2.0 framework". In: *the 2010 workshop*, 114–117. DOI=10.1145/2002333.2002351.

[5] C.-C. Chang and C.-J. Lin. 2011. "LIBSVM: A library for support vector machines." In: *ACM Transactions on Intelligent Systems and Technology* 2, 3, 27:1-27:27.

[6] L. Copeland and T. Gedeon. "The effect of subject familiarity on comprehension and eye movements during reading". In: *the 25th Australian Computer-Human Interaction Conference*, 285–288. DOI=10.1145/2541016.2541082.

[7] H. Drewes, R. Atterer, and A. Schmidt. 2007. "Detailed Monitoring of User's Gaze and Interaction to Improve Future e-Learning". In: *Proceedings of the 4th International Conference on Universal Access in Human-computer Interaction: Ambient Interaction*. UAHCI'07. Springer-Verlag, Berlin, Heidelberg, 802–811.

[8] H. Drewes, A. de Luca, and A. Schmidt. "Eye-gaze interaction for mobile phones". In: *the 4th international conference on mobile technology, applications, and systems and the 1st international symposium*, 364. DOI=10.1145/13780 63.1378122.

[9] M. L. Dybdal, J. S. Agustin, and J. P. Hansen. "Gaze input for mobile devices by dwell and gestures". In: *the Symposium*, 225. DOI=10.1145/2168556.2168601.

[10] A. Hyrskykari, P. Majaranta, A. Aaltonen, and K.-J. Räihä. "Design issues of iDICT". In: *the symposium*, 9–14. DOI=10.1145/355017.355019.

[11] R. J. K. Jacob. "What you look at is what you get: eye movement-based interaction techniques". In: *the SIGCHI conference*, 11–18. DOI=10.1145/97243.97246.

[12] M. Kumar. 2007. "GUIDe Saccade detection and smoothing algorithm". February 2007.

[13] M. Kumar, J. Klingner, R. Puranik, T. Winograd, and A. Paepcke. 2008. "Improving the Accuracy of Gaze Input for Interaction". In: *Proceedings of the 2008 Symposium on Eye Tracking Research & Applica tions*.

[14] O. Špakov. 2012. "Comparison of eye movement filters used in HCI". In: *Proceedings of ETRA'12 Symposium on Eye Tracking Research & Applications*.

[15] M. Porta. "Implementing eye-based user-aware e-learning". In: *Proceeding of the twenty-sixth annual CHI conference extended abstracts*, 3087. DOI=10.1145/1358628.1358812.

[16] D. Rozado, A. El Shoghri, and R. Jurdak. 2015. "Gaze dependant prefetching of web content to increase speed and

comfort of web browsing". In: *International Journal of Human-Computer Studies* 78, 31–42.

[17] S. Stellmach and R. Dachselt. "Investigating gaze-supported multimodal pan and zoom". In: *the Symposium*, 357. DOI=10.1145/2168556.2168636.

[18] V. Cantoni, C. J. Perez, M. Porta, and S. Ricotti. 2012. "Exploiting eye tracking in advanced e-learning systems". In: *Proceedings of the 13th International Conference on Computer Systems and Technologies*.

[19] C. Ware and H. H. Mikaelian. 1987. "An evaluation of an eye tracker as a device for computer input". *SIGCHI Bull.* 18, 4, 183–188.

[20] B. Wassermann, A. Hardt, and G. Zimmermann. 2012. "Generic Gaze Interaction Events for Web Browsers: Using the Eye Tracker as Input Device". In: *WWW2012 Workshop: Emerging Web Technologies, Facing the Future of Education*, 6.

[21] X. Zhang, X. Ren, and H. Zha. 2008. "Improving Eye Cursor's Stability for Eye Pointing Tasks". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. ACM, New York, NY, USA, 525–534. DOI=10.1145/1357054.1357139.

[22] ScienceCampus Tübingen. [Online] Available from: https://www.wissenschaftscampus-tuebingen.de/www/en/index.html?ref=folder5.

[23] K. Scheiter. *Knowledge Media Research Center Tübingen*. [Online] Available from: https://www.iwm-tuebingen.de/-www/en/mitarbeiter/ma.html?uid=kscheiter.

[24] R. Plötzner. *University of Education Freiburg*. [Online] Available from: https://www.ph-freiburg.de/imb/institut/in-stitutsleitung.html.

[25] A. Renkl. *University of Freiburg*. [Online] Available from: https://www.psychologie.uni-freiburg.de/Members/renkl.

[26] G. Zimmermann. *Stuttgart Media University*. [Online] Available from: https://www.hdm-stuttgart.de/Forschuung_transfer/forschungsschwerpunkte/responsive_media_experi-ence/team/zimmermann.

[27] E. Weisstein. 2016. [Online] Available from: http://math world.wolfram.com/LeastSquaresFitting.html