Fixation: A universal framework for experimental eye movement research*

Mostafa Elshamy
University of Southern California
Los Angeles, CA, USA
elshamy@alumni.usc.edu

Peter Khooshabeh
DEVCOM Army Research Laboratory
Los Angeles, CA, USA
peter.khooshabehadeh2.civ@mail.mil

ABSTRACT

We propose a free and open-source framework for experimental eye movement research, whereby a researcher can record and visualize gaze data and export it for analysis using an offline web application without proprietary components or licensing restrictions. The framework is agnostic to the source of the raw gaze stream and can be used with any eye tracking platform by mapping data to a standard JSON-based format and streaming it in real time. We leverage web technologies to address data privacy concerns and demonstrate support for recording at 300 Hz and real-time visualization.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; *Visualization*; • **Information systems** → *Web applications.*

KEYWORDS

eye tracking, experimental research, high-frequency recording, real-time visualization, web

ACM Reference Format:
Mostafa Elshamy and Peter Khooshabeh. 2021. Fixation: A universal framework for experimental eye movement research. In *ETRA '21: 2021 Symposium on Eye Tracking Research and Applications (ETRA '21 Short Papers), May 25–27, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3448018.3458007

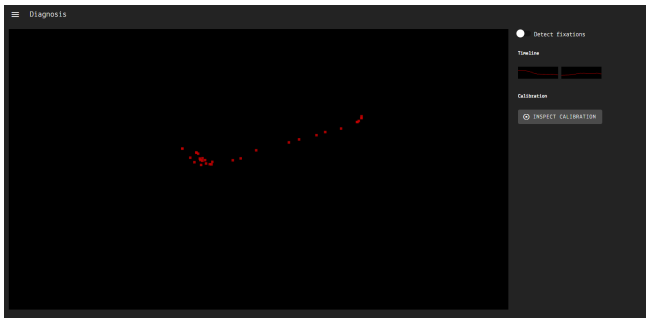

Figure 1: Fixation's main user interface. A switch labeled "Detect fixations" in the *diagnosis* module toggles between a trail of the last *n* raw gaze points and a scanpath of identified fixations visualized in real time.

1 INTRODUCTION

Eye tracking technology has advanced remarkably from its early days [Yarbus 1967] and is being used today to study computational models of vision [Itti and Baldi 2009], mutual gaze [Müller et al. 2020], whose role in social interaction was studied by psychologists decades earlier [Argyle and Cook 1976], and human–computer interaction [Nielsen and Pernice 2010].

While there has been significant progress in democratizing eye tracking [Krafka et al. 2016; Papoutsaki et al. 2016], commercial software coupled to expensive and specialized hardware remains dominant. Cost, licensing restrictions, and sparse documentation impede research progress. Tobii Pro Lab (formerly Tobii Pro Studio), SMI BeGaze (discontinued after Apple acquisition), and SR Research Experiment Builder/WebLink and Data Viewer are used most often to record and analyze eye tracking data [Blascheck et al. 2017]. Tobii Pro Sprint is a discontinued subscription-based cloud platform for user experience testing. Pupil Cloud is well documented, open-source software for head-mounted eye trackers [Kassner et al. 2014] and free within certain capacity caps.

Recent advances in machine learning and availability of large-scale datasets inspired a growing number of video-based gaze estimation models to appear in the literature [Valliappan et al. 2020; Zhang et al. 2019]. OpenGaze provides C++ APIs for gaze estimation using off-the-shelf cameras, but user-friendly software for conducting experimental research using such neural network models is not available yet.

We introduce novel methods to address problems of universal compatibility, scalable processing of high-frequency data, and data privacy. These methods are themselves quite interesting and generally applicable outside eye tracking to high-frequency recording and real-time visualization of neural or physiological signals.

2 ARCHITECTURE

Rather than overwhelm researchers with a complex feature set, we start with a basic set that we can grow based on community adoption and feedback. We provide high-performance modules for processing a raw gaze stream and fixation detection using a well-specified algorithm, capture and storage of stimulus and user camera frames during detected fixations and playback of stored frames overlaid with a scanpath or heatmap visualization.

*Fixation was developed under the codename "Spellbound," inspired by the dream sequence designed by Salvador Dali for the 1945 Alfred Hitchcock film of that title.

ETRA '21 Short Papers, May 25–27, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8345-5/21/05…$15.00
https://doi.org/10.1145/3448018.3458007

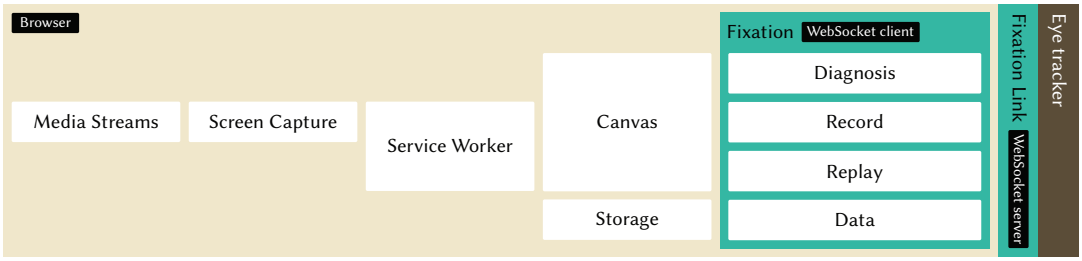

Figure 2: We reduce the problem of universal compatibility to mapping data before it is streamed over the network using *fixation links* to a client application running in-browser. Application modules use Web APIs to provide their functionality. e.g., the "Record" module uses the Canvas API for real-time visualization, the Service Worker API for offline cache storage, the Screen Capture API to capture the display as a media stream, and the Media Streams API to request access to a local webcam.

The idea of a web application was an early design decision to support distributed experiments, which proved useful for solving other problems (e.g., since we are able to capture HTML DOM structure during a fixation, we can analyze scanpaths without explicit area of interest specification as in previous literature [Eraslan et al. 2016]). However, there are 1) data privacy concerns in cloud environments due to the sensitivity of eye or face images and potential personal identification from gaze data [Liebling and Preibusch 2014] and 2) performance concerns when processing data at 60–300 Hz in real time, which can easily block the main thread in JavaScript. We show in Section 4 how to store sensitive data in the browser cache and in Sections 4 and 5 how a module can run scripts concurrently by spawning background threads known as *web workers*.

2.1 Fixation Link

A *fixation link* is a WebSocket[1] server that has a single function: transform the gaze stream into a standard JSON-based format (see Figure 5) and emit events to a connected WebSocket client so that it can correctly interpret gaze points. The `x` and `y` coordinates are normalized relative to the display such that $(0, 0)$ and $(1, 1)$ are the top-left and bottom-right corners respectively. The `timestamp` is measured in μs. Additional unstructured data (e.g., pupil size, eye position, or head pose in 3D space) may optionally be sent along and stored for analysis.

Since virtually all programming languages support JSON and WebSocket, it can be written in any programming language supported by an eye tracker's SDK or a gaze estimation model's API. We provide fixation links for Tobii TX300 (Python), Tobii EyeX (Node.js, C#, 48 sloc) and a simulator[2] for testing purposes (Python, 58 sloc).

3 DIAGNOSIS

A quick check before recording reduces the risk of lost or miscalibrated data. We provide visual tools to 1) verify the client is receiving WebSocket messages from a connected fixation link and can detect fixations and 2) inspect calibration accuracy to verify the absence of large errors.

[1]WebSocket is a protocol to enable web applications to maintain bidirectional communications with server-side processes.

[2]It is worth noting how the simulator works, which is by sampling gaze points from a Gaussian HMM with 9 hidden states (each corresponding to a screen region) trained on a small dataset of 27572 raw gaze points captured using a Tobii TX300.

Fixation detection is useful because it significantly reduces data size without compromising high-level analysis [Salvucci and Goldberg 2000]. We apply dispersion-threshold identification (I-DT) with dispersion threshold D_t and frequency f determined by a configurable duration threshold (often 100–200 ms). We start with an initial set of k gaze points $P = \{p_1, p_2, \ldots, p_k\} = \{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots, (x_k, y_k, t_k)\}$ where $k = D_t \times f$ and add additional points until the dispersion D exceeds the threshold D_t

$$D = [\max(X) - \min(X)] + [\max(Y) - \min(Y)] > D_t$$

where $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$ and $n \geq k$ then note a fixation at the centroid C

$$C = \frac{1}{n}\left(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i\right)$$

at timestamp t_1 and of duration $d = t_n - t_1$.

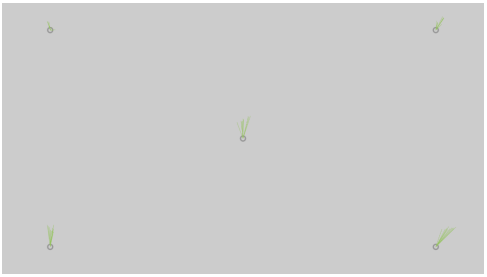

Figure 3: Visual inspection of mean absolute error indicates whether a re-calibration is needed.

Five-point calibration is generally adequate [Duchowski 2017], particularly if our goal is to determine if a re-calibration is needed before recording. We consider 5 points at $(0.1, 0.1)$, $(0.9, 0.1)$, $(0.5, 0.5)$, $(0.1, 0.9)$, $(0.9, 0.9)$ and plot horizontal and vertical mean absolute error E for n raw data points while gazing at p.

$$E = \frac{1}{n}\sum_{i=1}^{n} |x_i - x_1| + |y_i - y_1|$$

An early prototype provided a calibration tool that relied on an SDK-provided API. We observed that not all eye trackers support this, but almost all provide a free calibration tool.

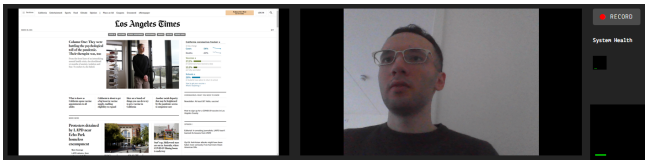

Figure 4: Left: Captured display media stream. Right: User camera stream. A visual indicator labeled "System Health" provides feedback about the status of the worker load. *Green* as opposed to *red* markers indicate that the system is adequately processing new frames.

4 RECORDING

Synchronization of independent data streams is a major problem in multimodal systems. If input data from different sources is recorded at different sampling rates, the clock of each source has to be synchronized with a master clock to reliably correlate the data. This has often been resolved by Network Time Protocol (NTP) based synchronization protocols [Mills 1991] or manual alignment. Recognizing that little to no visual processing is achieved during a saccade [Fuchs 1971], we adopt a different approach and capture stimulus and user camera frames at each identified fixation rather than record independent gaze, stimulus and user camera data streams. We believe this reduces problems of experimental reproducibility and results in more reliable datasets.

By capturing the screen as a media stream, we support any stimuli that can be displayed (e.g., static image, reading text on the web, or watching a film scene). To record the stimulus, we use the `getDisplayMedia()` method from the Screen Capture API. It returns a `MediaStream` object which can then be stored into a `video` element's `srcObject`. An `OffscreenCanvas` can use a `video` element as an image source and draw the image onto the canvas and we do this when we identify a fixation. A prespawned web worker then encodes the stimulus image, adds it to a JSON document representing the fixation and stores the augmented document (see Figure 5) in cache storage. A similar procedure using `getUserMedia()` from the Media Streams API can be used to record user camera frames.

```
{
  x: 0.305161,
  y: 0.560296,
  timestamp: 159391614.476313 // us
}
```

```
{
  id: 0,
  ...
  duration: 150.485869,
  stimulus: "..."
  user_cam: "..."
}
```

Figure 5: Sample JSON documents. Top: Raw gaze point. Bottom: An identified fixation, augmented with its duration and base64-encoded stimulus and user camera images.

We use a pool of n worker threads and for each new task select the worker W_{min} with minimum load $\min(\{l_0, l_1, \ldots, l_n\})$ where l_i is the load of worker i or the number of frames it is processing. This can be easily achieved by having each worker communicate its load back to the main thread after performing an asynchronous task. This is necessary to maintain high recording performance, and a naive synchronous approach will eventually block the main thread as encoding a frame is relatively slow and new frames will be generated at a rate faster than they can be encoded and stored in cache storage. We further reduce processing latency by never copying image data between execution contexts. *Transferable objects* can be used to transfer ownership of a byte array representing the image between the main thread and the web worker.

We considered both cache storage and IndexedDB for storing data. Both are widely supported, accessible from web workers, and asynchronous. IndexedDB can arguably be faster when seeking a frame during replay, but is a low-level API that requires significant setup before use. We show in Section 5 how to achieve fast seek performance using a frame buffer.

4.1 Tuning stimulus image encoding parameters

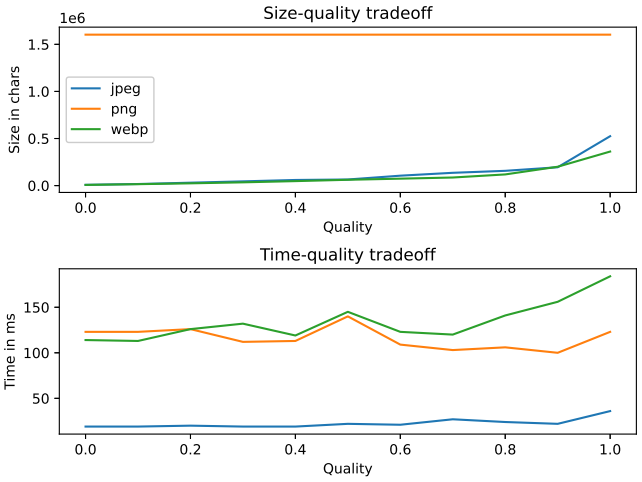

Figure 6: Trade-off between image quality and compression time and encoded image size using `canvas.toDataURL`.

Encoding a stimulus frame before storage significantly decreases storage requirements but can degrade image quality and recording performance if not done properly. We can optimize encoder options using a simple procedure: plot execution time and encoded image size as a function of image quality (using a 0.1 or 0.01 step) for every compression format, then visually inspect the plots for an optimal value. This can vary for different screen resolutions and stimuli. For example, we see from Figure 6 that the JPEG format is faster than both PNG and WebP, but a WebP encoder may produce fewer artifacts for graphical stimuli. The PNG format is lossless and results in significantly larger base64-encoded images.

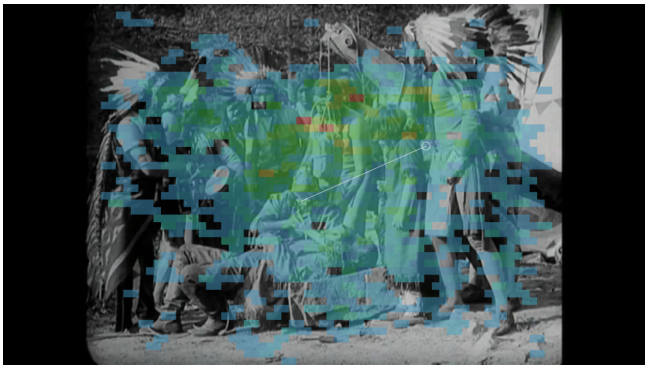

Figure 7: A dynamic heat map overlaid over the stimulus during replay of a recorded session. (Captured still from *White Fawn's Devotion* [Young Deer 1910]. Public Domain.)

5 REPLAY

Visualization of eye movements during replay of the recorded session is critical to verification of data integrity and exploration of potential pockets where further analysis may be useful or to running a retrospective think-aloud study. We provide two basic methods for gaze visualization: *scanpaths* and *heatmaps*. Both depict areas of relative cognitive importance, but scanpaths depict temporal viewing order while heatmaps depict aggregate gaze more intuitively.

We use a buffer of n frames and request the next n frames from a single prespawned web worker when we start consuming frames from the current buffer. A single worker consistently outperformed multiple workers in experiments to improve replay performance. We bypass decoding of base64-encoded images and instantly render frames on the canvas by creating n hidden elements in the document and setting their background image to the encoded image.

5.1 Rendering dynamic heatmaps

Rendering real-time heatmaps atop video is computationally intensive and has required a GPU implementation in the literature [Duchowski et al. 2012]. We extend an approach introduced in Biedert et al. [2012] to render discretized heatmaps in real time using a CPU implementation by relaxing the requirement for smooth rendering via a Gaussian point-spread function and taking advantage of eye tracking measurement error. It is an open question whether smooth heatmaps have advantages beyond visual familiarity, but we may consider a GPU-based option in future work.

If we consider a grid G of n tiles where each tile $t \in G$ has size $w_t \times h_t$ approximately equal to expected eye tracking error, then on every new gaze point $p = (x, y)$ we can check if $p \in t$ and increment i_t, intensity of tile t. A prespawned web worker is responsible for intensity accumulation. We can map the accumulated intensity count normalized relative to the maximum intensity i_{max} to a tile's alpha channel or to RGB color space via a color gradient. The traditional rainbow color map seen in Figure 7 can be generated through the `hsla()` functional notation such that *hot* has a hue of 209 and *cold* has a hue of 0 at 50% saturation, lightness, and alpha.

Finding the maximum intensity is the bottleneck of the CPU implementation and has $O(wh)$ complexity given a $w \times h$ frame, but it can be arbitrarily reduced by choosing the tile size. For example, choosing $w_t \approx \sqrt{w}$ and $h_t \approx \sqrt{h}$ reduces the search time to $O(\sqrt{wh})$.

Table 1: Storage requirements per session length

Duration	Frames	Usage
04:05	1305	50Mb (< 1%)
10:18	2746	111Mb (< 1%)
15:51	4171	184Mb (< 1%)

6 DATA MANAGEMENT

The data management module provides tools to import and export recorded data and information about cache content and disk space. Data can be exported as a JSON file and imported later by dragging-and-dropping it on a designated drop target. The exported document may be analyzed or used as a machine learning dataset.

How much data can be stored in the browser cache? Browser implementations vary, but the amount of available storage is usually a percentage of total disk space (e.g., Chrome and Firefox allow the browser to use up to 80% and 50% respectively). Table 1 shows usage and quota per session length as reported by the `estimate` method of the `StorageManager` interface of the Storage API in Chrome.

7 DISCUSSION

Existing open-source eye tracking software supports recording from hardware of a particular form factor (i.e., head-mounted) [Kassner et al. 2014], few popular models [Dalmaijer et al. 2014; Voßkühler et al. 2008], or focuses on post-experimental analysis of imported, prerecorded data [Kubler 2020]. Fixation is differentiated by its hardware-agnostic interface, inherently synchronized datasets, and accessibility on the web without compromising data privacy. We were able to experimentally verify our methods at 60–300 Hz, but they can in theory scale to higher sampling rates (e.g. 1000 Hz).

The development of Fixation was motivated by a limited budget during purchase of a Tobii TX300 eye tracker for an exploratory study of an eye movement phenomenon reported in Day [1964]. We were also concerned about unforeseen limitations of Tobii Pro Studio posing obstacles to our research. Commercial eye tracking software may process raw gaze data in partially specified pipelines to protect the manufacturer's intellectual property. We hope, by open-sourcing the framework and releasing it for free, that more researchers will have access to production-grade tools needed to obtain reliable and verifiable datasets. This promotes openness, decreases fragmentation within the research community, and accelerates experimental eye movement research.

Future work will involve benchmarking performance and growing the feature set to support use cases found to be problematic, but we are prioritizing some issues on our roadmap. We expect to 1) add more fixation links to support popular models and new models that appear in the literature, 2) provide additional event detection algorithms and methods for visualizing gaze data, and 3) explore using the platform to generate large datasets for training gaze estimation models. We encourage the eye tracking research community to contribute to the project.

ACKNOWLEDGMENTS

The authors thank Prof. Stacy Marsella, who acquired funding used to purchase the Tobii TX300 eye tracker. This work was supported in part by the U.S. Army RDECOM. Content does not necessarily reflect the position or policy of the U.S. government, and no official endorsement should be inferred.

CODE AVAILABILITY AND LICENSING

Fixation is hosted at https://polar-ocean-09884.herokuapp.com at the time of publication. Source code and documentation is available on the corresponding author's GitHub[3] under the MIT license.

REFERENCES

Michael Argyle and Mark Cook. 1976. Gaze and Mutual Gaze. (1976).

Ralf Biedert, Andreas Dengel, Mostafa Elshamy, and Georg Buscher. 2012. Towards Robust Gaze-Based Objective Quality Measures for Text. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (Santa Barbara, CA, March 28–30, 2012) *(ETRA '12)*. ACM, New York, NY, USA, 201–204. https://doi.org/10.1145/2168556.2168593

Tanja Blascheck, Kuno Kurzhals, Michael Raschke, Michael Burch, Daniel Weiskopf, and Thomas Ertl. 2017. Visualization of Eye Tracking Data: A Taxonomy and Survey. *Computer Graphics Forum* 36, 8 (Dec. 2017), 260–284. https://doi.org/10.1111/cgf.13079

Edwin S. Dalmaijer, Sebastiaan Mathôt, and Stefan Van der Stigchel. 2014. PyGaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. *Behavior Research Methods* 46, 4 (2014), 913–921. https://doi.org/10.3758/s13428-013-0422-2

Merle E. Day. 1964. An Eye Movement Phenomenon Relating to Attention, Thought and Anxiety. *Perceptual and Motor Skills* 19, 2 (Oct. 1964), 443–446. https://doi.org/10.2466/pms.1964.19.2.443

Andrew T. Duchowski. 2017. *Eye Tracking Methodology: Theory and Practice.* Springer, Cham, Switzerland. https://doi.org/10.1007/978-3-319-57883-5

Andrew T. Duchowski, Margaux M. Price, Miriah Meyer, and Pilar Orero. 2012. Aggregate Gaze Visualization with Real-Time Heatmaps. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (Santa Barbara, CA, March 28-30, 2012) *(ETRA '12)*. ACM, New York, NY, USA, 13–20. https://doi.org/10.1145/2168556.2168558

Sukru Eraslan, Yeliz Yesilada, and Simon Harper. 2016. Eye Tracking Scanpath Analysis Techniques on Web Pages: A Survey, Evaluation and Comparison. *Journal of Eye Movement Research* 9, 1 (Feb. 2016). https://doi.org/10.16910/jemr.9.1.2

Albert F. Fuchs. 1971. The Saccadic System. In *The Control of Eye Movements*, Paul Bach-y Rita, Carter C. Collins, and Jane E. Hyde (Eds.). Academic Press, New York, NY, USA, 343–362. https://doi.org/10.1016/B978-0-12-071050-8.50017-3

Laurent Itti and Pierre Baldi. 2009. Bayesian surprise attracts human attention. *Vision Research* 49, 10 (June 2009), 1295–1306. https://doi.org/10.1016/j.visres.2008.09.007

Moritz Kassner, William Patera, and Andreas Bulling. 2014. Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-based Interaction. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Seattle, WA, September 13–17, 2014) *(UbiComp '14 Adjunct)*. ACM, New York, NY, USA, 1151–1160. https://doi.org/10.1145/2638728.2641695

Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, and Antonio Torralba. 2016. Eye Tracking for Everyone. In *IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV, June 26–30, 2020) *(CVPR '16)*. IEEE Computer Society, Las Alamitos, CA, USA, 2176–2184. https://doi.org/10.1109/CVPR.2016.239

Thomas C. Kubler. 2020. The Perception Engineer's Toolkit for Eye-Tracking data analysis. In *ACM Symposium on Eye Tracking Research and Applications* (Stuttgart, Germany, June 2–5, 2020) *(ETRA '20 Short Papers)*. ACM, New York, NY, USA, Article 15, 4 pages. https://doi.org/10.1145/3379156.3391366

Daniel J. Liebling and Sören Preibusch. 2014. Privacy Considerations for a Pervasive Eye Tracking World. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Seattle, WA, September 13–17, 2014) *(UbiComp '14 Adjunct)*. ACM, New York, NY, USA, 1169–1177. https://doi.org/10.1145/2638728.2641688

David L. Mills. 1991. Internet Time Synchronization: The Network Time Protocol. *IEEE Transactions on Communications* 39, 10 (1991), 1482–1493. https://doi.org/10.1109/26.103043

Philipp Müller, Ekta Sood, and Andreas Bulling. 2020. Anticipating Averted Gaze in Dyadic Interactions. In *ACM Symposium on Eye Tracking Research and Applications* (Stuttgart, Germany, June 2–5, 2020) *(ETRA '20 Full Papers)*. ACM, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3379155.3391332

Jakob Nielsen and Kara Pernice. 2010. *Eyetracking Web Usability.* New Riders, Berkeley, CA, USA.

Alexandra Papoutsaki, Patsorn Sangkloy, James Laskey, Nediyana Daskalova, Jeff Huang, and James Hays. 2016. WebGazer: Scalable Webcam Eye Tracking Using User Interactions. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence* (New York, NY, July 9–15, 2016) *(IJCAI '16)*. AAAI, Palo Alto, CA, USA, 3839–3845.

Dario D. Salvucci and Joseph H. Goldberg. 2000. Identifying Fixations and Saccades in Eye-Tracking Protocols. In *Proceedings of the 2000 Symposium on Eye Tracking Research and Applications* (Palm Beach Gardens, FL, November 6–8, 2000) *(ETRA '00)*. ACM, New York, NY, USA, 71–78. https://doi.org/10.1145/355017.355028

Nachiappan Valliappan, Na Dai, Ethan Steinberg, Junfeng He, Kantwon Rogers, Venky Ramachandran, Pingmei Xu, Mina Shojaeizadeh, Li Guo, Kai Kohlhoff, et al. 2020. Accelerating Eye Movement Research via Accurate and Affordable Smartphone Eye Tracking. *Nature Communications* 11, 1, Article 4553 (Sept. 2020), 12 pages. https://doi.org/10.1038/s41467-020-18360-5

Adrian Voßkühler, Volkhard Nordmeier, Lars Kuchinke, and Arthur M. Jacobs. 2008. OGAMA (Open Gaze and Mouse Analyzer): open-source software designed to analyze eye and mouse movements in slideshow study designs. *Behavior Research Methods* 40, 4 (2008), 1150–1162. https://doi.org/10.3758/BRM.40.4.1150

Alfred L. Yarbus. 1967. *Eye Movements and Vision.* Springer, New York, NY, USA. https://doi.org/10.1007/978-1-4899-5379-7

James Young Deer. 1910. White Fawn's Devotion. Retrieved March 31, 2021 from https://www.filmpreservation.org/preserved-films/screening-room/t1-white-fawn-s-devotion-1910

Xucong Zhang, Yusuke Sugano, and Andreas Bulling. 2019. Evaluation of Appearance-Based Methods and Implications for Gaze-Based Applications. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, UK, May 4–9, 2020) *(CHI '19)*. ACM, New York, NY, USA, Article 416, 13 pages. https://doi.org/10.1145/3290605.3300646

[3]https://github.com/melhosseiny/fixation