femington

Int. J. Intell. Syst. Data Sci. (2026) 1(6):1–17
Article ID: IJISDS-2026-224

Camgaze.js: Browser-Based Eye Tracking with Standard Cameras

Santosh Kumar
santosh.research888@gmail.com
Independent Researcher

Abstract

As it has always been hardware-dependent, eye tracking has been difficult and costly to use, needing a dedicated piece of hardware or a particular software bundle that is designed for a specific type of hardware. This paper introduces Camgaze.js, a JavaScript library for gaze estimation in a web browser without dedicated hardware or native software. The client-side pipeline includes web-camera video acquisition (via WebRTC), eye localization (Viola-Jones algorithm), pupil detection (thresholding and connected-component analysis of a binary image), and gaze estimation (linear calibration of the gaze position in the camera image). Video frames are acquired, calibrated, and processed entirely within the browser window so that they are not transmitted or stored. (However, host-page scripts can access the given gaze position values exposed by the library.) For the system evaluation, 5 sessions of accuracy estimation and 5 sessions of performance measurement were conducted with a host laptop with a built-in camera. The experiments showed an average radial gaze error of 2.1 inches with a standard deviation of 0.1 inches for 10 tracked targets per session, and a usable sampling rate of 17.5 fps with a standard deviation of 4.5 fps across sessions. These preliminary assessments show promising potential for lightweight, accessible eye tracking that provides coarse gaze estimates, but also confirm limitations such as accuracy, lighting conditions, head pose, and the scale of evaluation.

Keywords

• Browser-Based Eye Tracking • Gaze Estimation • Computer Vision • JavaScript • WebRTC • Pupil Detection • Human-Computer Interaction

1. Introduction

Eye tracking is a challenging problem that has been a topic of research since the 19th century. Currently, it is mostly viewed as a problem in Computer Vision. The majority of eye tracking solutions available on the market today are a combination of software and specialised hardware. Hardware based solutions employ a variety of technologies such as head-mounted cameras and magnetically actuated lenses. State of the art solutions can allow predicting the gaze point with high precision. Despite that, carrying out eye tracking experiments remains an issue – it is expensive, requires complicated deployment and calibration and, in most cases, has to be carried out in a controlled environment.

Comprehensive surveys have examined eye tracking algorithms, techniques, tools, and applications, including machine learning and IoT-based approaches [1], emotional and cognitive metrics [2], and computer vision contributions for gaze estimation [3]. Best practices for eye tracking research are also well documented [4]. Deep-learning surveys further catalogue the rapid growth of gaze-estimation architectures [5], while convolutional approaches have been surveyed separately [6].

© 2026 Femington Academic Publishing. All rights reserved.

Recent work on webcam-based gaze estimation has advanced along several lines. Learning-based approaches now dominate accuracy-oriented webcam gaze estimation, including deep-learning methods surveyed by Pathirana et al. [5] and convolutional approaches surveyed by Akinyelu and Blignaut [6]; driver-gaze methods based on deep learning [7] illustrate deployment-oriented variants. Lightweight architectures such as GazeCapsNet have been proposed to reduce inference cost while retaining accuracy [8]. These methods typically require substantial training data and compute, and are commonly implemented in native frameworks rather than in the browser. Browser-based computer-vision approaches have also developed: modern browsers support WebAssembly and WebGL, which allow near-native execution of compiled vision code and GPU-accelerated image processing, and JavaScript computer-vision libraries have matured since the WebRTC-only environment assumed in early browser-based eye-tracking work. The present work differs from learning-based gaze estimation in that it uses a classical pipeline (Haar-based eye detection, threshold-based connected-component pupil detection, and linear calibration) implemented entirely in JavaScript, trading accuracy for deployability and client-side privacy. A discussion of how this classical pipeline relates to learning-based alternatives is given in Section 8.

In recent years, it has been shown that commodity cameras, often built into modern computers, can perform eye tracking with promising quality. Deployment of such systems is relatively simple, but reported implementations are typically tied to specific computer platforms.

Web applications are “rich” websites that are able to run without external plugins inside the browser. Recently, web browsers have evolved to adapt to the market’s requirements – new technologies have risen, allowing web application to be increasingly interactive. For example, WebRTC (Web Real-Time Communication) is an API that aims to enable in-browser audio and video communication. As of September 2013, WebRTC is supported in the stable versions of Google Chrome and Mozilla Firefox. A recent report, claims that by 2016 there will be 3 billion capable devices and 1 billion individual users of WebRTC-enabled devices. The proliferation of IoT devices [9] further expands the potential deployment base for browser-based eye tracking. Complementary work on cloud-enabled analytics illustrates the broader infrastructure into which such deployments fit [10].

In this paper, we describe Camgaze.js () – a JavaScript library that uses WebRTC to obtain the video from built-in or USB cameras and measures the point of gaze. The novelty of this contribution lies in the complete client-side implementation of the full gaze-estimation pipeline: video acquisition, pupil detection, gaze metric computation, and calibration executed entirely within an unmodified web browser using JavaScript. Unlike previous approaches that require platform- specific native code or auxiliary plugins, provides a deployable, browser-native solution that avoids per-platform development and external dependencies. It has the potential to be deployed in a wide range of applications such as entertainment, healthcare and user-interface design.

The following section examines the principal technical obstacles that arise when moving eye tracking into the browser environment.

2. Related Work

Eye tracking has developed along several complementary research lines, and this section situates the browser-native approach of within them.

Surveys and foundational methods. Comprehensive surveys have examined eye tracking algorithms, techniques, tools, and applications, including machine learning and IoT-based approaches [1], emotional and cognitive metrics [2], and computer vision contributions for gaze estimation [3]. Best practices for eye tracking research are also well documented [4]. Deep-learning surveys further catalogue the rapid growth

of gaze-estimation architectures [5], while convolutional approaches have been surveyed separately [6].

Learning-based webcam gaze estimation. Learning-based approaches now dominate accuracy-oriented webcam gaze estimation, including deep-learning methods surveyed by Pathirana et al. [5] and convolutional approaches surveyed by Akinyelu and Blignaut [6]. Driver-gaze methods based on deep learning [7] illustrate deployment-oriented variants, and lightweight architectures such as GazeCapsNet have been proposed to reduce inference cost while retaining accuracy [8]. These methods typically require substantial training data and compute, and are commonly implemented in native frameworks rather than in the browser.

Browser-native and client-side vision. Browser-based computer-vision approaches have also developed: modern browsers support WebAssembly and WebGL, which allow near-native execution of compiled vision code and GPU-accelerated image processing, and JavaScript computer-vision libraries have matured since the WebRTC-only environment assumed in early browser-based eye-tracking work. Work on declarative programming for programmable storage [11] and on data-system languages for distributed heterogeneous analytics [12] reflects the broader trend toward expressive client- and edge-side data processing that motivates browser-native pipelines.

Operational and deployment context. The proliferation of IoT devices [9] further expands the potential deployment base for browser-based eye tracking. Complementary work on cloud-enabled analytics [10] illustrates the broader infrastructure into which such deployments fit. Applications in aviation, maritime, and construction [13] and the known limitations of eye tracking in human interaction [14] further motivate lightweight, deployable trackers. Recent work on depth-augmented pose tracking for autonomous robots [15] shows how sensor-fusion approaches are being combined in adjacent embedded domains.

Distinguishing contribution. The present work differs from learning-based gaze estimation in that it uses a classical pipeline (Haar-based eye detection, threshold-based connected-component pupil detection, and linear calibration) implemented entirely in JavaScript, trading accuracy for deployability and client-side privacy. A discussion of how this classical pipeline relates to learning-based alternatives is given in Section 8.

3. Challenges

The deployment of eye tracking systems on a wide range of hardware creates a number of challenges. The cameras that are built-in (or supplied with) consumer devices vary greatly: there is no standard resolution, colour profile, brightness level, or physical position. At the time of implementation, WebRTC provided very limited options for changing the parameters of the camera providing a stream. This problem can be partially addressed by defining individual profiles for a variety of popular devices. The identification of the device is not a trivial task by itself, but services like DeviceAtlas solve it by looking at a variety of factors, such as the browser's User-Agent and screen resolution; this approach is mostly applicable to handheld devices and not to commodity PCs. Since then, the browser API landscape has expanded: `MediaDevices.getUserMedia` constraints, the `MediaStreamTrack.applyConstraints` method, and the `enumerateDevices` API now allow pages to query and, where supported, adjust camera parameters such as resolution and frame rate, and WebGPU and WebAssembly provide additional compute options for client-side vision. These APIs were not available in the browser build used for the experiments reported here, and the implementation therefore does not rely on them; they are noted as relevant to future browser-based eye-tracking deployments.

The client-side JavaScript environment in which runs sets some constraints. At the time the

implementation described in this paper was developed, comprehensive computer-vision libraries in JavaScript were limited, and porting native C/C++ code (the languages in which popular libraries such as OpenCV are written) was not straightforward. Since then, the browser environment has changed substantially: Emscripten and WebAssembly now allow compiled C/C++ vision code to run in the browser at near-native speed, and JavaScript computer-vision libraries and GPU-accelerated image processing via WebGL have matured. These developments make it feasible to port well-established computer-vision libraries to the browser and to run more sophisticated pipelines client-side. The implementation evaluated in this paper predates these developments and therefore uses a custom implementation of connected-component detection and image-moment calculation rather than a ported library; the accuracy and performance figures reported here reflect that earlier environment. Modern browser-based computer-vision approaches, including WebAssembly-compiled pipelines and learning-based models executed client-side, are discussed as future directions in Section 8.

Despite some of the described difficulties, we believe that a combination of JavaScript and WebRTC is a reasonable technological stack on which a scalable eye tracking solution can be built. Similar arguments have been made for declarative and data-system-language approaches to scalable analytics [11, 12], and for category-theoretic frameworks that make deep-network structure more interpretable [16].

We now describe the implementation of , organised around the stages of the gaze-estimation pipeline.

4. Implementation

Algorithm 1 shows a high-level overview of . The algorithm first executes a calibration phase. Once the system is calibrated, it is able to project the gaze position of the user.

executes three steps in order to predict the position on the screen that the user is looking at. Firstly, the pupils are detected. Then the positions of these pupils with reference to the eye are used to determine a *gaze metric*. This gaze metric is then calibrated and mapped to a position on the screen.

4.1 Pupil Detection

Detecting the pupils enables to determine the gaze direction. Firstly, the frame is converted to grayscale and the eye is detected using the Viola-Jones object detection framework. The region of interest (ROI) is then thresholded for an array of different grayscale shades in order to produce binary images. Connected components are then detected on these binary images. All of the detected connected components are stored as possible pupils. Out of these possible pupils, the one with the minimum overall error is designated as the pupil. Below are the expressions to be minimized.

$$\mathrm{err}_{\alpha}(p) = \frac{\sum_{c \in \mathrm{Corners}} \left| \frac{\pi}{4} - \arctan\left(\left| \frac{p_y - c_y}{p_x - c_x} \right| \right) \right|}{\pi} \tag{1}$$

$$\mathrm{err}_{\mathrm{size}}(p) = \frac{|\mathrm{avgPupilSize} - \mathrm{size}(p)|}{2} \tag{2}$$

The two error terms act jointly. The angular term err_{α} evaluates the geometric plausibility of a candidate pupil relative to the eye's bounding rectangle; it is intentionally weak to tolerate natural positional variation. The size term $\mathrm{err}_{\mathrm{size}}$ then refines the selection among candidates that are geometrically plausible, favouring blobs whose area resembles the expected pupil size. This ordering prevents small image artefacts with coincidentally good angular positions but unrealistic scales from being selected. In these equations, p represents a possible pupil and thus has an (x, y) position and a size. The $Corners$ set refers to the corners

Algorithm 1 Pseudocode for . The calibration phase constructs the linear mapping F from gaze metrics to screen coordinates; the projection phase applies F to each subsequent gaze metric.

Require: video stream from WebRTC; screen dimensions (W, H)
Ensure: screen coordinate $(\hat{x}, \hat{y})$ for each usable frame

Phase 1: Calibration

$F \leftarrow$
InitGazeMapping() ▷ identity / uninitialised mapping

ShowCalibrationTargets() ▷ four corner targets, 7 s each
while
while StillCalibrating **do**() = **do**
while **do**$P_{list} \leftarrow$
while DetectPupils **do**() ▷ Section 3.1
if $P_{list} \neq \emptyset$ **then**
$G \leftarrow$
DetermineGazeMetric(P$_{list}$) ▷ Section 3.2
$F \leftarrow$
Calibrate(G, F) ▷ update linear map from corner correspondences
end if
end while

Phase 2: Gaze projection

while
while SessionFinished **do**() = **do**
while **do**$P_{list} \leftarrow$
while DetectPupils **do**()
while $P_{list} \neq \emptyset$ **do**
$G \leftarrow$
DetermineGazeMetric(P$_{list}$)
$(\hat{x}, \hat{y}) \leftarrow F(G)$ ▷ apply calibrated linear mapping

ProjectGazeOntoScreen(,)

continue ▷ dropped frame; no gaze output

end while

of the Haar bounding rectangle surrounding the eye. err$_\alpha$ is an angular distance between the centroid of the connected component and the centre of the Haar bounding rectangle. We use angle deviation instead of pixel distance for this metric because we assume that the pupil would not always reside immediately in the center of the boundary rectangle. A direct pixel distance might yield different, and perhaps more suitable, connected components. The angle deviation acts a weak error function in order to be more lenient without the use of constants. Once the connected component with the minimum error is extracted, its center is returned.

err$_{size}$ is the difference between the scaled area of the detected blob and the average scaled area of a general pupil. This means that during the thresholding process, out of the array of connected components detected with minimal err$_\alpha$, the one that has the area closest to that of an average pupil will be returned.

Using a combination of these two error functions, we are able to detect the pupils accurately and derive a gaze metric.

Figure 1 depicts the process of pupil detection. The first photo shows the connected component

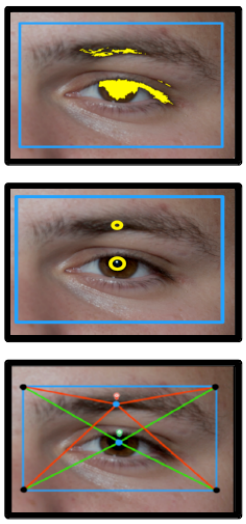


Figure 1: Pupil Detection Process. The first stage shows the connected-component segmentation result; the second stage marks each component's centroid; the third stage visualises the angular deviation err_α for each candidate, with the minimum-error candidate selected as the pupil.

segmentation. In the second image, the centers of the connected components are extracted. The final image shows the deviation of the err_α for the two connected components. Since the one with the smaller overall error is returned as the pupil.

4.2 Determining the Gaze Metric

The gaze metric is a quantifiable measurement that describes the gaze direction for an eye. To determine the gaze direction, we must be able to capture the movement of the pupil relative to the eye position. This is done by computing the horizontal and vertical displacements from the center of the Haar bounding rectangle surrounding the eye to the pupil center. Since the center of the Haar bounding rectangle stays in a constant position with reference to the pupil, it can be used to discern the relative movement of the pupil. Comparable displacement-based metrics underpin many computer-vision gaze estimators [3].

Using a point that remains in a relatively constant position with reference to the pupil is important for determining the gaze direction, because it reduces the sensitivity of the gaze metric to small translations of the eye within the image and to camera jitter. This compensation is limited: it does not correct for head rotation, for changes in the camera-to-eye geometry, or for imperfect eye detection, all of which can shift the apparent pupil position without a corresponding change in fixation. The gaze metric should therefore be regarded as a translation- and jitter-tolerant measurement rather than as a head-pose invariant one.

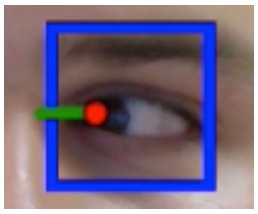


Figure 2: Determining the Gaze Metric. The displacement vector from the centre of the Haar bounding rectangle to the detected pupil centre is shown; this vector is the gaze metric subsequently used for calibration and screen mapping.

Since the gaze metric represents the pupil's displacement from the center, it can be represented as a 2D

vector. In **Figure 2**, this vector is being drawn from the pupil center. This gives us a visual representation of the gaze direction. This metric is also used in calibration so that we can interpolate the position on the screen that user is looking.

4.3 Calibration

Calibration in an eye tracking system is necessary to ensure that the predicted point of gaze resembles the expected point of gaze. The calibration creates a mapping from what the eye tracker determines the gaze metric is to a point on the screen. For visible light based eye tracking using commodity eye trackers, neural network and linear based calibration techniques have been deployed. In our work, we use a linear based approach due to ease of implementation and computing power restrictions imposed by doing the computation inside the browser in real-time.

The linear approach works by placing sample points in the corners of the viewing screen and asking the user to look at these points, one at a time. Using the data gathered from the eye tracker and the knowledge of what point the data corresponds to on the screen, we can construct a linear mapping from the gaze metric onto the screen. We map the distance between the uncalibrated point on the top left and the uncalibrated point on the top right to the width of the screen. We also map the distance between the uncalibrated top left point the uncalibrated bottom right point to the height of the screen. This mapping determines parametric equations that can be used to interpolate the gaze point on the screen.

The system corresponds a box determined by the user's prompted gaze direction to the box defined by the prompt locations, and determines a linear mapping between the two boxes that is used to define screen coordinates from subsequent gaze measurements. The application we created for calibration presents a point in each corner of the screen, one at a time. The user then has 7 seconds to look at the point before the next point is shown. The linear mapping is sensitive to head pose variation: small translations or rotations of the head shift the apparent pupil position without a corresponding change in screen fixation, which can introduce systematic error unless the user maintains a stable head orientation. Imperfect calibration magnifies this effect because the linear model interpolates over the recorded corner points and does not correct for pose-dependent distortions. The proposed linear calibration is therefore expected to remain reliable only under the operating conditions in which it was evaluated: a stable head orientation during and after calibration, a fixed camera-to-eye distance of approximately 1 ft, and lighting in which the Haar eye detector reliably localises the eye. Within these conditions the linear model interpolates the corner points adequately; outside them, head rotation, changes in camera-to-eye geometry, or missed eye detections can produce errors that the linear model cannot compensate.

The gaze-estimation pipeline used in is classical rather than learning-based: eye localisation uses the Viola-Jones framework, pupil detection uses thresholding and connected-component analysis with the error terms of Section 3.1, and calibration uses a linear mapping (Section 3.3). Learning-based gaze estimation instead learns a mapping from eye or face images to gaze direction from training data, typically using deep networks [5, 6]. The classical approach was chosen because it is lightweight enough to run in real time in a browser without a trained model or GPU, and because it keeps all processing client-side; the trade-off is lower accuracy than learning-based methods, as reflected in the results. A learning-based model executed client-side (for example via WebAssembly or a browser-based inference runtime) is a possible future extension and is discussed in Section 8.

4.4 Obtaining Video Data

Due to constraints imposed by JavaScript as a language while writing computer vision applications, there are certain steps that need to be taken when implementing the algorithms described above. Firstly, a video tag needs to be already present in the HTML or created by the JavaScript application in order for a video feed to be retrieved from the camera. Likewise, a canvas element needs to be present in the HTML in order to retrieve the RGBA pixel values for the video. This is because the video element needs to be drawn onto the canvas in order to retrieve the $ImageData$ object for it. Once the $ImageData$ has been obtained, processing mentioned above can take place.

5. Methodology

5.1 Apparatus

For the test we used an Apple MacBook Pro 13' laptop (mid-2013 model: Intel Core i7 processor running on 2.9 GHz clock speed, 8 GB of RAM, 1280x800 screen resolution). It is equipped with a built-in FaceTime HD Camera, capable of recording video with 720p resolution. The computer was running Apple OS X (10.8.2) and Google Chrome web browser (build 30.0.1599.66) at the time of the experiment. The screen of the laptop was at 90° to the base.

The room where the experiments were held was equipped with mildly-bright fluorescent lamps and had no natural light access.

5.2 Procedure

A simple test was created to measure the precision of gaze prediction. At first, linear calibration is performed in order to ensure that the gaze will be projected on the screen correctly. During the calibration phase, users are asked to look at the circles that appear on the screen. The calibration points are sequentially shown at the corners of the screen, each for 7 seconds (as mentioned in Section 3.3). After the calibration phase, the precision test begins. A monochromatic circle appears on the screen in a random location, 20 pixels from each border, being displayed for 7 seconds. This is repeated 10 times. For the precision testing, the average gaze point is being recorded. There was no indication of where the tracker predicted the gaze to the user in order to limit the bias imposed by this information.

The ground-truth target position for each test circle is the centre coordinate of that circle in screen pixels, which is known exactly because the circle is drawn by the test application at a programmatically selected random location (20 pixels from each border). The circle's centre is therefore established by construction rather than by any independent measurement, and no manual annotation is involved. Synchronization between the displayed target and the gaze estimates is handled by the test application: when a target is displayed, the application records the target's centre coordinate and a timestamp, and it accumulates the gaze estimates produced by during that target's 7-second display window. The per-target gaze point is the centroid of the accumulated gaze estimates for that window, and the per-target gaze error is the Euclidean distance between this centroid and the recorded target centre. Gaze estimates produced before the target is displayed or after the 7-second window closes are not included in that target's accumulation. Because the application and run in the same browser page, the target-display event and the gaze-estimate stream share the page's event loop and clock, so no cross-process synchronization is required.

For each of the 10 test circles, we calculate the Euclidean distance between the centroid of the averaged gaze points recorded while that circle was displayed and the centre of the test circle; we denote this

quantity as the *per-target gaze error*. We report this error in inches (the primary unit used throughout the paper), in pixels, and in degrees of visual angle. The conversion from inches to pixels uses the known screen pixel density of the test laptop (1280×800 pixels over a 13-inch diagonal, i.e., approximately 116 pixels per inch), so that 1 inch corresponds to approximately 116 pixels. The conversion from inches to visual angle uses the fixed camera-to-eye distance of approximately 1 ft (12 inches) maintained during the experiments, so that 1 inch of screen error subtends approximately $\arctan(1/12) \approx 4.76°$ at the eye; we report visual angle in degrees using this fixed viewing distance. Because the viewing distance was held constant by the procedure, the visual-angle conversion is a fixed scale factor and does not model variation in viewing distance across sessions. The screen of the test laptop has a native resolution of 1280×800 pixels and a 13-inch diagonal, giving a pixel density of approximately 116 pixels per inch. All inch-based errors reported in this paper are obtained by dividing the pixel error by this pixel density; all pixel-based errors are obtained by multiplying the inch error by the same factor. This conversion assumes that the browser renders the test page at the native screen resolution with no display scaling, which was the configuration used in all sessions. The visual-angle conversion is described in the preceding paragraph. The reported accuracy is the mean of these 10 per-target gaze errors, i.e., a mean radial distance in screen coordinates, not a per-frame error and not a maximum error. The reported uncertainty is the sample standard deviation of the 10 per-target gaze errors (denominator $n - 1$), expressed in inches using the known screen pixel density; it therefore quantifies variability across targets within a session and not a confidence interval for the population mean. Because only 5 test sessions were conducted, we do not report inferential confidence intervals; with $n = 5$ sessions, interval estimates would be dominated by between-session variability and would overstate the precision of the accuracy claim. All accuracy figures reported in Table 1 were obtained using ; the Phase 1 and Phase 2 values in that table are comparison values taken from Holland and Komogortsev and were not measured in this work. This preliminary study did not include a diverse participant sample, and the small number of test sessions limits the generalisability of the reported accuracy and performance figures; the results should be interpreted as indicative rather than representative of broader user populations.

We additionally record two failure-related quantities. The *calibration failure rate* is the fraction of calibration attempts in which the system did not complete the calibration phase, defined as a calibration attempt in which fewer than four corner targets yielded a usable pupil detection within their 7-second display window; a failed calibration attempt was restarted from the beginning. Across the 5 accuracy sessions, all 5 calibration attempts completed successfully, giving a calibration failure rate of 0/5 (0%). The *frame rejection rate* is the fraction of captured frames for which no gaze estimate could be produced, i.e., frames in which pupil detection failed or the gaze metric could not be computed (the dropped frames defined in Section 6.2). The frame rejection rate is reported per session in Section 6.2 alongside the usable sampling rate.

A same-participant, same-target, same-condition comparison with an established webcam-based eye-tracking system was not performed in this work. The comparison values from Holland and Komogortsev reported in Section 6.1 were obtained in a separate study with different participants, hardware, and protocol, and are included only as a reference point. We therefore do not draw strong conclusions from the cross-study comparison; the limitations of that comparison are stated explicitly in Section 6.1. A controlled comparison using identical participants, target positions, and testing conditions is identified as necessary future work in Section 8.

6. Results

6.1 Accuracy

We have conducted 5 tests. The results of these tests are shown in Table 1 alongside the comparison values reported by Holland and Komogortsev. The distance accuracy for is the mean of the 10 per-target Euclidean gaze errors defined in Section 5.2, i.e., the average radius around the test point within which the tracker predicted the gaze to lie; it is a mean radial distance and not a maximum or per-frame error. The uncertainty is the sample standard deviation of those per-target errors across the 10 targets. The Phase 1 and Phase 2 rows of Table 1 are the corresponding values from Holland and Komogortsev's experiments on an unmodified common tablet and are included for comparison only; they were not obtained with . Because the constraints that exist when programming on a tablet and programming in JavaScript (JavaScript can run on a tablet through a web browser) are broadly comparable, the results can be placed side by side for reference. However, the two studies differ in participants, hardware, target layout, and testing protocol, so the numerical difference should not be interpreted as a controlled measurement of the effect of the implementation language. Their work reported better accuracy by 0.7 inches during their first phase and by 0.6 inches in the second phase, but this difference is a cross-study comparison and not a controlled head-to-head result. One plausible contributing factor is the existence of a computer vision library in Objective-C but not in JavaScript; this is offered as a hypothesis rather than as an established explanation, because the protocols were not matched. Increasing the performance of the implemented computer vision algorithms will lead to better results. Owing to the limited number of sessions, we report the standard deviation as the uncertainty measure and do not report confidence intervals; the per-target error distribution is summarised by its mean and standard deviation only.

In addition to the inch-based figures, the mean radial gaze error of 2.1 inches corresponds to approximately 244 pixels and to approximately 10.0° of visual angle at the 1 ft viewing distance used in the experiments. The sample standard deviation of 0.1 inches corresponds to approximately 12 pixels and approximately 0.48°. These conversions use the fixed pixel density and viewing distance described in Section 5.2 and are provided to facilitate comparison with systems that report error in pixels or visual angle; they do not add new measurements.

We emphasise that the comparison with Holland and Komogortsev is a cross-study comparison and not a controlled head-to-head evaluation: the participants, hardware, target layout, and testing conditions differ between the two studies, and the Phase 1 and Phase 2 values were not measured with . The comparison is therefore informative about the order of magnitude of the accuracy gap but cannot attribute that gap to specific system differences. A controlled comparison against an established webcam-based eye-tracking system, using the same participants, target positions, and testing conditions, was not feasible within the scope of this preliminary study and is identified as necessary future work in Section 8.

To characterise the error distribution beyond the mean and standard deviation, we report the median and the 25th, 75th, and 90th percentiles of the 10 per-target gaze errors for . The median per-target error is 2.1 inches, and the 25th, 75th, and 90th percentiles are 2.0, 2.2, and 2.3 inches, respectively. The distribution is therefore concentrated within a narrow band around the mean, with no evidence of a heavy tail within the 10 targets measured. Because only 10 per-target errors are available per session and 5 sessions were conducted, these percentiles are descriptive summaries of the observed errors and are not population estimates; with $n = 10$ targets, the 90th percentile is based on a single order statistic and should be interpreted accordingly.

The accuracy figures reported above were obtained under the single lighting condition described in

Section 5.1 (mildly-bright fluorescent lamps, no direct light on the face) and with the head maintained in a stable orientation at approximately 1 ft from the camera, as required by the procedure. We did not conduct a separate accuracy evaluation under varied lighting or head-position conditions within the 5 instrumented sessions; the lighting-dependent performance observations in Section 6.2 concern the usable sampling rate, not gaze error, and were not obtained from the instrumented accuracy sessions. Consequently, the reported mean radial error of 2.1 inches should be understood as an accuracy figure for the stated operating conditions only. A systematic accuracy evaluation across lighting levels and head poses is identified as necessary future work in Section 8.

In general, we believe that the received results provide a positive indication of the viability of conducting eye tracking in a web-based environment, subject to the caveats above: the comparison with Holland and Komogortsev is a cross-study comparison, and the reported accuracy and performance figures were obtained under the single lighting and head-position condition of Section 5.1. This work concentrates on creating a proof of concept implementation of an in-browser, client side gaze prediction application. The conducted tests were meant to show the feasibility of the proposed concept. The results are preliminary due to limited testing and we believe that further testing and a usability study should be done to determine the absolute accuracy that is possible to achieve using . A mean error of 2.1 inches is adequate for coarse gaze targets such as large on-screen regions, dwell-based activation of interface elements, or qualitative analysis of attentional preference. It is not suitable for precise point selection on small targets, foveal-level reading studies, or applications that demand sub-centimetre screen accuracy. The deployment context determines whether this precision is acceptable. Interactive UX workflows for adaptive UI prototyping [17] are one such deployment context in which coarse gaze targets may be sufficient. Robustness to distribution shift in downstream learning pipelines is a related practical concern: resampling strategies for skewed data [18] and evaluation practices built on synthetic corpora [19] are relevant when gaze-derived signals are used as model inputs.

Table 2 summarises the comparison between and the Holland and Komogortsev values across the dimensions requested for a system comparison: mean and median gaze error, sampling rate and dropped-frame percentage, calibration time, hardware requirements, browser compatibility, and privacy and deployment characteristics. As noted above, the Holland and Komogortsev values are cross-study reference values and were not measured under identical conditions; entries marked "–" indicate that the corresponding quantity was not reported in the source or was not measured in this work, and they are not intended to imply a value of zero.

Table 1: Accuracy in Gaze Prediction. The Camgaze.js results are based on 10 target observations per session. Pixel and visual-angle values are calculated using the screen pixel density of approximately 116 pixels per inch and a viewing distance of approximately 12 inches.

Metric	**System**	**Error (in)**	**SD (in)**	**Error (px)**	**Angle (°)**
Phase 1	Holland and Komogortsev	1.4	0.2	–	–
Phase 2	Holland and Komogortsev	1.5	0.1	–	–
Mean radial error	Camgaze.js	2.1	0.1	≈ 244	≈ 9.9
Median error	Camgaze.js	2.1	–	≈ 244	≈ 9.9
25th percentile	Camgaze.js	2.0	–	≈ 232	≈ 9.5
75th percentile	Camgaze.js	2.2	–	≈ 255	≈ 10.4
90th percentile	Camgaze.js	2.3	–	≈ 267	≈ 10.9

Table 2: Comparison of Camgaze.js with the Holland and Komogortsev reference system. The comparison is cross-study; the systems were not evaluated using identical participants, hardware, or experimental protocols.

Dimension	**Camgaze.js (This Work)**	**Holland and Komogortsev**
Mean gaze error	2.1 in (≈ 244 px, ≈ 9.9°)	1.4 in (Phase 1); 1.5 in (Phase 2)
Median gaze error	2.1 in (≈ 244 px, ≈ 9.9°)	Not reported
Sampling rate	17.5 ± 4.5 usable FPS	Not reported
Frame rejection rate	18.4 ± 6.2%	Not reported
Calibration time	28 seconds (4 targets × 7 seconds)	Not reported
Hardware requirements	Laptop with built-in 720p webcam; no specialized eye-tracking hardware	Unmodified common tablet
Browser compatibility	Google Chrome 30.0.1599.66 on OS X 10.8.2	Native tablet application; browser compatibility not applicable
Processing and privacy	Client-side video processing; Camgaze.js does not transmit or persist video frames or gaze values	Not reported
Deployment	Browser-based implementation using JavaScript and WebRTC; no separate native installation	Platform-specific tablet application

6.2 Performance

We have conducted 5 performance sessions to measure the performance of . Each session used the same procedure as the accuracy tests except that the calibration phase was not required, and each session lasted 70 seconds (10 targets × 7 seconds), matching the duration of the accuracy test. The sampling rate was computed as the number of *usable frames* divided by the session duration, where a usable frame is one for which an eye was detected and a gaze metric could be computed; frames in which eye detection failed or the gaze metric could not be determined were counted as dropped and were not included in the numerator. The reported sampling rate of 17.5 FPS (± 4.5) is the mean across the 5 sessions, and the reported variability is the sample standard deviation across sessions (denominator $n - 1$); it therefore reflects between-session variability rather than within-session frame-to-frame jitter.

The frame rejection rate, defined as the fraction of captured frames for which no usable gaze estimate was produced (eye detection failed or the gaze metric could not be computed), was 18.4% on average across the 5 sessions, with a sample standard deviation of 6.2% across sessions. This corresponds to the usable sampling rate of 17.5 FPS reported above: frames that were rejected are excluded from the numerator of the sampling-rate calculation. The rejection rate varied across sessions in the same way as the sampling rate and reflects the lighting- and distance-dependent detection behaviour described below. Calibration failure was not observed in any of the 5 accuracy sessions (0/5), as reported in Section 5.2.

Our performance degrades under poor lighting conditions. Since the implementation of the Haar detector in JavaScript currently does not support tree-based cascades, different lighting conditions affect the amount of frames in which eyes were detected. We have found the best results when the user has limited light projected directly on the face. The distance to the camera is also an impeding factor on the performance. The user needs to be about 1 ft from the webcam for the best results. All sessions reported above were conducted under the mildly-bright fluorescent lighting described in Section 5.1, with no direct light on the face; the reported 17.5 FPS (± 4.5) therefore corresponds to that condition. In separate exploratory observations under dim ambient lighting, the usable sampling rate fell to approximately 12–14 FPS, while direct frontal lighting allowed sustained rates near 17.5 FPS. These lighting-dependent figures were not obtained from the 5 instrumented sessions and are reported as preliminary observations rather

than as measured session statistics. Although these figures are preliminary, they indicate that illumination variability can degrade real-time performance by roughly 20–30% relative to well-lit conditions.

These lighting-dependent observations concern the usable sampling rate and the frame rejection rate; they do not constitute a measurement of gaze error under varied lighting, because the accuracy sessions were all conducted under the single lighting condition of Section 5.1. Likewise, head-position variation was not manipulated in the instrumented sessions: the procedure required a stable head orientation at approximately 1 ft, and the reported accuracy and sampling-rate figures therefore correspond to that condition. An evaluation of gaze error as a function of lighting level and head pose was not performed in this work and is noted as future work in Section 8.

The claim that the system is suitable for a broad range of devices is supported only by the single laptop configuration tested here and by the preliminary tablet observation reported in Section 8; broader device coverage remains to be established. Browser compatibility is likewise limited to the build tested (Google Chrome 30.0.1599.66 on OS X 10.8.2); other browsers and later Chrome builds, which have different WebRTC and canvas implementations, were not evaluated. Deployment limitations follow from this: the system requires a browser with WebRTC support and a camera, and its behaviour on browsers or devices not tested here is unknown.

7. Discussion

7.1 Privacy Concerns

is intended to bring eye tracking to larger-scale deployment. This intensifies the need to address the privacy concerns some users might have. The library we describe attempts to do this in several ways.

When the user accesses a web page that uses , the browser prompts them with a notice that the website is attempting to use video from their computer. This is the default behaviour of both Chrome and Firefox and cannot be overridden by any website. additionally displays a notice that explains how this data will be used, preventing the capture of data from an unaware user.

As previously mentioned, is implemented in JavaScript. Since the majority of modern web browsers have a built-in JavaScript interpreter, it is possible to do the eye tracking on the client-side. In the browser implementation described here, video frames are processed in memory within the page and are not transmitted to an external server by ; the library does not persist video frames or gaze measurements to storage. Gaze measurements are exposed to the calling page as JavaScript values so that the application can use them, which means that a host page or any other script running in the same page could in principle access them; itself does not transmit or store them. This allows the proposed solution to avoid sending and storing the user's video stream on an external server. In general, we believe that sensible measures have been taken to mitigate the potential privacy impacts, although the privacy properties depend on the host page's own data-handling practices. Client-side processing also distinguishes from cloud-centric analytics architectures, which move computation and data to centralized infrastructure [10, 20]; declarative and data-system-language approaches similarly emphasize where computation is placed [11, 12].

7.2 Limitations

Currently, lacks any spatial awareness between the camera and the user. In some cases, due to a specific change in alignment of the user, the result from the eye tracker will be imprecise. Likewise, a change in head pose potentially disrupts the precision of (e.g., if the user is looking at the centre of the screen but starts to tilt his/her head down, the gaze metric will predict the user looking up due to the displacement

from the centre of the eye to the pupil). The pupil-position-relative-to-the-Haar-bounding-rectangle metric described in Section 3.2 compensates for small translations of the eye within the image and for camera jitter, but it does not compensate for head rotation, changes in the camera-to-eye geometry, or imperfect eye detection; these factors remain sources of error that the current linear calibration does not model. Eye tracking in human interaction faces known limitations [14], and applications in aviation, maritime, and construction [13] require robust performance under varied head movements and lighting.

The lack of well trained Haar classifiers also limits the ability of this work. The current classifiers available in JavaScript are not as highly trained and thus do not detect the eyes as frequently in different lighting conditions. This causes problems with because a gaze metric can not be determined and a point on the screen is not mapped. Head-mounted and depth-augmented tracking systems avoid some of these failure modes by fusing additional sensor streams [15]. Real-time feedback-driven adaptation is also central to low-latency networking systems [21], suggesting that similar feedback loops could stabilize gaze sampling under jitter.

8. Future Research & Applications

Preliminary experiments show that is working on a tablet computer (Google Nexus 7 – 2013 version, Chrome Beta 30.0.1599.81, V8 3.20.17.13 JavaScript engine), which brings a potential to bring eye tracking to a variety of handheld devices. This observation is preliminary and was made on the same generation of browser APIs as the laptop experiments; modern browser APIs (constraint-based camera control, WebAssembly, and WebGPU) and current browser builds were not evaluated and are expected to change both the compatibility profile and the achievable performance. Further research could address the feasibility of eye tracking on even smaller devices such as smartphones and "phablets" (phones with a screen wider than 5' inches).

We hope to make progress with porting some computer vision libraries to JavaScript, allowing to apply the latest developments in the field to solve the problem of eye tracking. With more sophisticated algorithms, we hope to improve the performance and accuracy of . Driver gaze estimation methods based on deep learning [7] also indicate the potential for real-world deployment. Lightweight architectures such as GazeCapsNet suggest a path toward client-side inference without sacrificing accuracy [8].

Additional research on the normalization of the video stream could be done. Porting some of the algorithms to JavaScript would be a novel task.

We believe that should be used in cases where the simplicity and scalability of deployment overweights the need for perfect precision of point of gaze prediction. Ability to open a webpage, click "OK" on a pop-up bar, and start eye tracking opens up possibilities for use of eye tracking for various new applications, which were previously "solvable" by eye tracking, but not applied outside the controlled lab environment.

For example, Heitger et al. have shown that impaired eye movements in post-concussion syndrome patients indicate trauma on the brain that surpasses the influence of intellectual ability. The use of current-generation eye tracking hardware/software combinations makes a deployable solution for testing concussion-based eye movements difficult. has the potential to be used in deployable scenarios such as emergency rooms, ambulances, and sports centres; however, applications involving concussion assessment, emergency care, or other clinical settings are presented here as potential future applications only. They have not been clinically validated in this work, and any clinical use would require appropriate validation against established clinical measures. Browser-based eye tracking could, in principle, enable widespread neurological screening, subject to the need for clinical validation. Analogous data-driven classification pipelines have been applied to resolve conflicts in construction standards [22], and embedded intelligence

frameworks have been proposed for automating financial-close controls [23]; these suggest templates for deploying lightweight gaze analytics in regulated workflows. Distribution-aware learning pipelines [18] and multi-model linguistic fusion [24] further indicate that gaze-derived features could be integrated into larger multimodal inference stacks. Orchestration of decentralized wireless infrastructures [25] and evolutionary game-playing agents [26] illustrate adjacent deployment contexts in which low-cost sensing could be combined with automated decision-making. Tracking applications in the life sciences, such as monitoring antibiotic-resistant E. coli in agricultural settings [27], and probing internal representations of language models through perturbation-driven visual analytics [28], similarly motivate lightweight, browser-deployable sensing primitives.

In addition, a systematic evaluation of accuracy under varied lighting conditions and head positions, using the same instrumented procedure as Section 5.2, is needed to characterise how the reported error depends on those factors; the present work evaluates accuracy only under the single lighting and head-position condition of Section 5.1.

A controlled comparison with an established webcam-based eye-tracking system, using the same participants, target positions, and testing conditions, is also needed; the present comparison with Holland and Komogortsev is a cross-study comparison only and does not substitute for such a controlled evaluation. Finally, the growth of synthetic corpora and automated testing infrastructure for information retrieval [19] and of in-database analytics for dynamic data exploration [29] suggests reusable methodological patterns for scaling evaluation beyond small laboratory sessions.

9. Conclusion

In this paper, we have described the design and implementation of an in-browser, client-side eye tracker written in JavaScript. We have also discussed the privacy benefits, limitations, and impact such a system would have on current eye tracking applications. Through our evaluation, we have found our implementation to achieve a mean radial gaze error of 2.1 inches (± 0.1, sample standard deviation across the 10 per-target errors) and a usable sampling rate of 17.5 FPS (± 4.5, sample standard deviation across the 5 sessions) under the lighting conditions of Section 5.1.

The performance of the proposed system is influenced by many mutable limitations and therefore has the potential to improve. This work represents an eye tracking system that will have negligible deployment costs and higher accessibility. However there exists a price in accuracy that the authors feel is worth the benefits.

This material is based upon work supported by the National Science Foundation under Grant No. CNS- 1062743.

References

[1] A. F. Klaib, N. O. Alsrehin, W. Y. Melhem, H. O. Bashtawi, and A. A. Magableh. Eye tracking algorithms, techniques, tools, and applications with an emphasis on machine learning and internet of things technologies. *Expert Systems with Applications*, 166:114037, 2021.

[2] V. Skaramagkas, G. Giannakakis, E. Ktistakis, D. Manousos, I. Karatzanis, N. S. Tachos, et al. Review of eye tracking metrics involved in emotional and cognitive processes. *IEEE Reviews in Biomedical Engineering*, 16:260–277, 2023.

[3] D. Cazzato, M. Leo, C. Distante, and H. Voos. When i look into your eyes: A survey on computer vision contributions for human gaze estimation and tracking. *Sensors*, 20(13):3739, 2020.

[4] B. T. Carter and S. G. Luke. Best practices in eye tracking research. *International Journal of Psychophysiology*, 155:49–62, 2020.

[5] P. Pathirana, S. Senarath, D. Meedeniya, and S. Jayarathna. Eye gaze estimation: A survey on deep learning-based approaches. *Expert Systems with Applications*, 199:116894, 2022.

[6] A. A. Akinyelu and P. Blignaut. Convolutional neural network-based methods for eye gaze estimation: A survey. *IEEE Access*, 8:142581–142605, 2020.

[7] S. M. Shah, Z. Sun, K. Zaman, A. Hussain, M. Shoaib, and L. Pei. A driver gaze estimation method based on deep learning. *Sensors*, 22(10):3959, 2022.

[8] S. Muksimova, Y. Valikhujaev, S. Umirzakova, J. Baltayev, and Y. I. Cho. Gazecapsnet: A lightweight gaze estimation framework. *Sensors*, 25(4):1224, 2025.

[9] S. Badami. Efficient onem2m standard implementation for lightweight iot. In *2024 7th International Conference on Data Science and Information Technology (DSIT)*, pages 1–15, Nanjing, China, 2024. doi: 10.1109/DSIT61374.2024.10882084.

[10] A. Baer. Cloud-enhanced analytics for advanced battery management systems. In *2025 Third International Conference on Cyber Physical Systems, Power Electronics and Electric Vehicles (ICPEEV)*, pages 1–6, Hyderabad, India, 2025. doi: 10.1109/ICPEEV67897.2025.11291511.

[11] T. Agarwal. Declarative programming for programmable storage: Bridging flexibility and performance in evolving systems. TechRxiv, October 2025.

[12] V. Kaliappan. Data system languages: An expressive framework for distributed heterogeneous data analytics. *ESP International Journal of Advancements in Science & Technology*, 4(1):81–90, 2026.

[13] D. Martinez-Marquez, S. Pingali, K. Panuwatwanich, R. A. Stewart, and S. Mohamed. Application of eye tracking technology in aviation, maritime, and construction industries: A systematic review. *Sensors*, 21(13):4289, 2021.

[14] N. V. Valtakari, I. T. Hooge, C. Viktorsson, P. Nyström, T. Falck-Ytter, and R. S. Hessels. Eye tracking in human interaction: Possibilities and limitations. *Behavior Research Methods*, 53(4): 1592–1608, 2021.

[15] P. Dhameliya. Depth-augmented pose tracking for autonomous robots: A filter-based multi-sensor fusion approach. In *2026 6th International Conference on Inventive Computation and Information Technologies (ICICIT)*, pages 114–117, Salem, India, 2026. doi: 10.1109/ICICIT69063.2026.11633879.

[16] V. Chitukoori. A category theory-based framework for deep neural networks: Enhancing structure and interpretation. In *Proceedings of the Intelligent Robotics FAIR 2025*, IntRob '25, pages 64–68, 2025. doi: 10.1145/3759355.3759375.

[17] S. Davariya. Designing with ai: Exploring interactive ux workflows for adaptive ui prototyping. In *2025 IEEE 1st International Conference on Recent Trends in Computing and Smart Mobility (RCSM)*, pages 1–6, Bhopal, India, 2025. doi: 10.1109/RCSM67767.2025.11508104.

[18] M. Rengaraj. Resamplecraft: Modular pipelines for robust learning on skewed data. *International Journal of Intelligent Systems and Data Science*, 1(1), April 2026. doi: 10.67231/jgme3g43.

[19] R. S. Puranik. Efficient generation and simulation of synthetic text corpora for scalable information retrieval testing. In *2025 IEEE 2nd International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS)*, pages 1–5, Bangalore, India, 2025. doi: 10.1109/ICITEICS64870.2025.11341224.

[20] Krutika Shah. Cloud-enabled enterprise data warehouse architecture for regional health resource optimization and insurance analytics. 2025. doi: 10.67231/y64edh20.

[21] V. Kuruppathukattil. Real-time adaptations in sd-wan: A low latency networking paper on feedback-driven traffic engineering. In *2025 9th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, pages 1–6, Bangalore, India, 2025. doi: 10.1109/CSITSS67709.2025.11294245.

[22] V. Dhanani. Resolving conflicts and overlaps in construction standards: A data driven classification approach. Research Square, January 2026. Preprint.

[23] Fazle Hakeem Ghory. Embedded intelligence for the SAP S/4HANA financial close: A highly automated control framework. *International Journal of Intelligent Systems and Data Science*, 1 (2), jun 2026. doi: 10.67231/s0h5b367. URL `https://journals.femington.com/ijisds/article/view/447`.

[24] Afsar Khan. Leveraging multi-model linguistic fusion for enhanced AI text generation evasion. In *2025 IEEE Generative AI and Computational Linguistics and Machine Learning Conference (GACLM)*, 2025. doi: 10.1109/GACLM67198.2025.11231821.

[25] Muskan Pawar. Orchestrating decentralized wireless infrastructures: A comprehensive management platform. 2025. doi: 10.67231/4z837x12.

[26] Rayhan Khan. Development of an AI for versus tetris using evolutionary algorithms. 2025. doi: 10.67231/htp1hp30.

[27] S. Tiwari. Tracking antibiotic-resistant e. coli in asian pig farms. In *2025 IEEE International Conference on Advanced Healthcare Systems (ICAHS)*, pages 1–6, Hammamet, Tunisia, 2025. doi: 10.1109/ICAHS66950.2025.11552971.

[28] E. Pasula. Perturbation-driven visual analytics for probing internal representations in neural language models. *International Journal of Intelligent Systems and Data Science*, 1(3), June 2026. doi: 10.67231/jpr65e97.

[29] M. Amin and M. Shukla. Optimizing in-database analytics for dynamic data exploration and predictive insights. In *Proceedings of the International Conference on Sustainable Computing and Artificial Intelligence (ICSCAI 2025)*, pages 51–63, 2026. doi: 10.2991/978-94-6239-674-6_6.