Camgaze.js : Mobile Eye Tracking and Gaze Prediction in JavaScript

Alex Wallar
Christian Poellabauer
Patrick Flynn

Abstract

Eye tracking is a difficult problem that is usually solved using specialised hardware and therefore has limited availability due to cost and deployment difficulties. We describe Camgaze.js a client-side Javascript library that is able to measure the point of gaze using only commodity optical cameras. In our work, we conduct experiments using Camgaze.js to show the usability of such a system. We also discuss the challenges and applications of using an in browser eye tracking system. Since the described eye tracker works inside the browser without any additional installation setup, it provides a more feasible scope of use.

1 Introduction

Eye tracking is a challenging problem, that has been attempted to be solved since the 18th century (Ahrens, A., 1891. Die Bewegung der Augen beim Schreiben. Rostock: University of Rostock). Currently, it is mostly viewed as a proble in Computer Vision. The majority of eye tracking solutions available on the market today are a combination of software and specialised hardware. The principles behind hardware varies greatly: it ranges from head-mounted cameras to lenses with integrated coils. It is believed that the state of the art solutions can allow accuracy up to NN% link. 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.

In the recent years, it has been shown (Agustin, 2009; Sewell and Komogortsev, 2010) that it is possible to use commodity cameras, often built into modern computers to perform eye tracking with promising quality. Deployment of such systems is relatively simple, but in the described cases it is tied to specific computer platforms [Holland and Komogortsev 2012].

Web applications are rich websites that are able to run without external plugins inside the browser.

2 Challenges

3 Implementation

`Camgaze.js` goes through two steps in order to predict the gaze direction. Firstly, `Camgaze.js` detects each pupil. It then uses the pupils deviation from a unique point on the face to determine the gaze metric, $\mathcal{G}$. This metric needs to be calibrated in order for there to be a mapping from $\mathcal{G}$ to a point on the screen. Once this gaze metric has been calibrated, `Camgaze.js` should be able to interpolate area of the screen the user is looking at. A high level description of the algortihm is shown below.

Algorithm 1 Pseudocode for `Camgaze.js`

$\mathcal{F} \leftarrow$ InitGazeMapping$()$
while StillCalibrating() $==$ `true` **do**
  $P_{list} \leftarrow$ DetectPupils()
  $\mathcal{G} \leftarrow$ DetermineGazeMetric(P_{list})
  $\mathcal{F} \leftarrow$ Calibrate$(\mathcal{G}, \mathcal{F})$
while SessionFinished() $==$ `false` **do**
  $P_{list} \leftarrow$ DetectPupils()
  $\mathcal{G} \leftarrow$ DetermineGazeMetric(P_{list})
  ProjectGazeOntoScreen$(\mathcal{F}(\mathcal{G}))$

3.1 Pupil Detection

Detecting the pupils enables `Camgaze.js` to determine the gaze direction. Pupil detection in this approach is aimed to be fast in order to be deployable onto mobile devices. Firstly, the frame is converted to grayscale and the eye is detected using the Viola-Jones Object Detection Framework [Viola and Jones 2001]. The region of interest (ROI) is then thresholded for an array of different colors and blob detection takes place. 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.

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

$$\textsc{err}_{size}(p) = \frac{|\textit{avgPupilSize} - \textsc{size}(p)|}{2} \quad (2)$$

$\textsc{err}_{\alpha}$ refers to how far the blob center is from the center 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 to close to the center and a direct pixel distance might yield other blobs more suitable. The angle deviation acts a weak error function in order to be more lenient without the use of constants. Once the blob with the minimum error is extracted, the center of the blob is returned.

3.2 Determining the Gaze Metric

The gaze metric is determined by establishing a reference point that will in a constant position with reference to the pupil center. Using this point, we are able to capture the motion of the pupil without the influence of head movement or tablet jitter.

4 Methodology

5 Applications

6 Discussion

7 Limitations

8 Future Research

References

Holland, C., and Komogortsev, O. V. 2012. *Eye Track-*

ing on Unmodified Common Tablets: Challenges and Solutions, In Proceedings of ACM Eye Tracking Research & Applications Symposium, Santa Barbara, CA.

VIOLA, P., AND JONES, M. 2001. Robust real-time object detection. In *International Journal of Computer Vision*.