EyeDroid: An Open Source Mobile Gaze Tracker on Android for Eyewear Computers

Shahram Jalaliniya
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen S, Denmark
jsha@itu.dk

Daniel Garcia Garcia
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen S, Denmark
dgac@itu.dk

Diako Mardanbegi
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen S, Denmark
dima@itu.dk

Ioannis Sintos
IT University of Copenhagen
Rued Langgaards Vej 7
2300 Copenhagen S, Denmark
isin@itu.dk

Ubicomp/ISWC '15 Adjunct, September 07-11, 2015, Osaka, Japan
©2015 ACM. ISBN 978-1-4503-3575-1/15/09. . . $15.00
DOI: http://dx.doi.org/10.1145/2800835.2804336

Abstract

In this paper we report on development and evaluation of a video-based mobile gaze tracker for eyewear computers. Unlike most of the previous work, our system performs all its processing workload on an Android device and sends the coordinates of the gaze point to an eyewear device through wireless connection. We propose a lightweight software architecture for Android to increase the efficiency of image processing needed for eye tracking. The evaluation of the system indicated an accuracy of 1.06 degrees and a battery lifetime of approximate 4.5 hours.

Author Keywords

Gaze tracking; Eyewear computer; Android; Google Glass

ACM Classification Keywords

H.5.2. [Information interfaces and presentation: User Interfaces]: Input devices and strategies

Introduction

By emerging new generation of unobtrusive eyewear computers, such as Google Glass[1] and Vuzix smart glass[2], it seems feasible that eventually these eyewear devices play role in everyday tasks. Due to the special form factor of eyewear devices, the delay between intention and action

[1] $https://developers.google.com/glass/$
[2] $http://www.vuzix.com/consumer/products_m100/$

is very short compared to other mobile devices[10]. This opens new opportunities for eyewear computers to be used more on the move and in parallel with real world tasks. However, mobile interaction with eyewear devices is still challenging. For example, in a mobile scenario, sometimes hands of the user are busy with a manual activity, or the user might be doing a visually demanding task. This means the eyewear device needs to provide several channels for interaction to support users in different situations. State of the art eyewear devices already support for head gesture input, voice commands, and touch-based gestures. Eye gaze has also been studied as an input modality for head-mounted displays [3]. However, due to the technical limitations, gaze-based interaction is not still supported by state of the art eyewear devices. One of the main challenges is the fact that the image processing required for gaze tracking is extremely complex and power demanding. Unfortunately, this computational demand is very far from what can be accomplished on existing eyewear devices such as Google Glass. In this paper, we investigate the possibility of using an Android smartphone to process eye images captured by a head-mounted camera to calculate the gaze coordinates for an eyewear computer.

Related Work

Most of the recent mobile gaze trackers use a laptop in user's backpack [5, 6] or a remote computer [9, 1] to analyze the eye image and calculate gaze coordinate. The dependency of gaze tracking systems to a local or remote computer decreases the mobility of users. There are also some commercial products from companies such as EyeTribe[3], Tobii[4], and Umoove[5] which support eye tracking on mobile and wearable devices. But the commercial mobile gaze trackers are usually so expensive and hard to afford. That is the reason why some of the recent studies have tried to use cheap and small processors such as Raspberry Pi [2] and micro-controllers [7] for eye tracking. Ferhat et.at [2] have presented a cheap eye tracking solution running on a Raspberry Pi device. They based their work on the open source Opengazer [8]. The average gaze estimation error of their system is about $1.4°$ for an image size of 640 × 480 pixels with the frame rate of 3Hz. Although their system was running on a small device, it was only tested on a stationary setup for gaze tracking on a computer screen. A more relevant work to our paper is the iShadow eye tracker by Mayberry et.al [7] that focuses on head-mounted gaze tracking. They have presented a fully mobile eye tracking solution using a low-power ARM Cortex M3 micro-controller.

[3] *https://theeyetribe.com/*
[4] *http://www.tobii.com/*
[5] *http://www.umoove.me/*

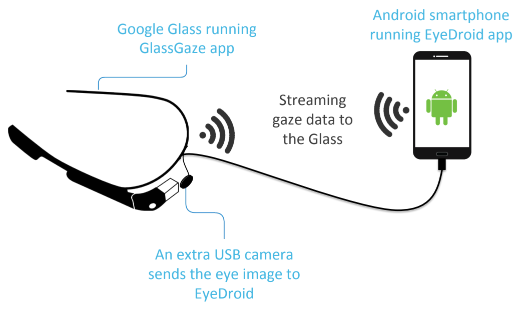
Google Glass running GlassGaze app
Android smartphone running EyeDroid app
Streaming gaze data to the Glass
An extra USB camera sends the eye image to EyeDroid

Figure 1: A schematic view of the system Architecture.

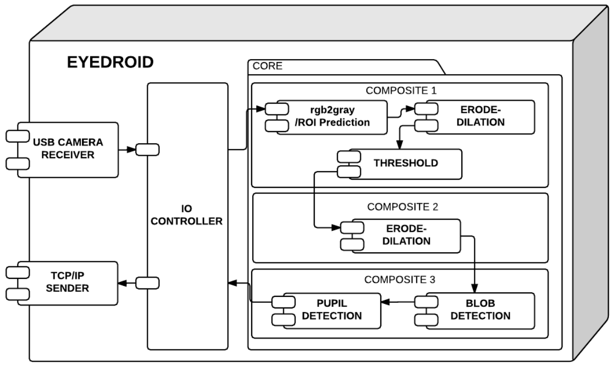

Figure 2: EyeDroid software arcgitecture. Eye tracking algorithm inside the core is decomposed into steps (filters) and connected by pipes (arrows). Each composite is executed on a separate thread.

The focus of the iShadow system was mainly implementing a very efficient video-based eye tracking approach that can run on a small micro-processor. They achieved real-time gaze tracking in an image captured by a front-view camera and they have reported an error of about 3 degrees for their system. Since in eyewear computers the display size is very small (less than 15 degrees), the accuracy of the eye tracker should be higher to provide a graceful interaction. In this paper, to achieve a higher accuracy in eye tracking (about 1 degree), we rely on the processing capacity of commonly used mobile devices. The proposed eye tracker on mobile device is an open-source affordable solution for gaze tracking on eyewear computers.

System Architecture

The proposed system comprises two main components: (1) our gaze tracking application (EyeDroid) running as a server on an android smartphone, and (2) the client application on Google Glass (GlassGaze)[6]. A schematic view of the system architecture is represented in Figure 1.

EyeDroid: Gaze Tracker Server on Android

Hardware

The hardware requirements in the current implementation of the EyeDroid eye tracker are an Android mobile device (minimum API level 15) and a head mounted USB 2.0 infrared camera connected directly to the Android phone through a USB cable. The first hardware prototype of the EyeDroid is shown in Figure 5. The recommended camera resolution is 640×480 pixels. Because the Android platform does not provide support to connect an external USB camera, the operating system needs to own root access to the phone and use customized camera video drivers. To develop the EyeDroid gaze tracker, open source third party drivers are used [4].

Software Architecture

The software architecture of the EyeDroid application is designed based on pipes and filters design pattern. This architecture helped us test different algorithm configurations easily during system development. Also in the EyeDroid software platform we built and used Java Lightweight Processing Framework (JLPF)[7] (Figure 3) as an external library. This design allows for a fully configurable algorithm in terms of decomposability and scheduling of the steps for execution on the available processing resources, instead of a monolithic algorithm that would perform poorly. Finally, in order to divide the algorithm in steps of equal execution time, the composite pattern was implemented to allow composition of individual steps (see Figure 2). Since performance is a

[6] $https://github.com/centosGit/GlassGaze$
[7] $https://github.com/centosGit/JLPF$

critical issue, we used the Android NDK support for C++ instead of the regular Android SDK for java. This allowed the algorithm code to run directly on the processing resources and access system libraries directly, unlike Java which would run on a virtual machine.

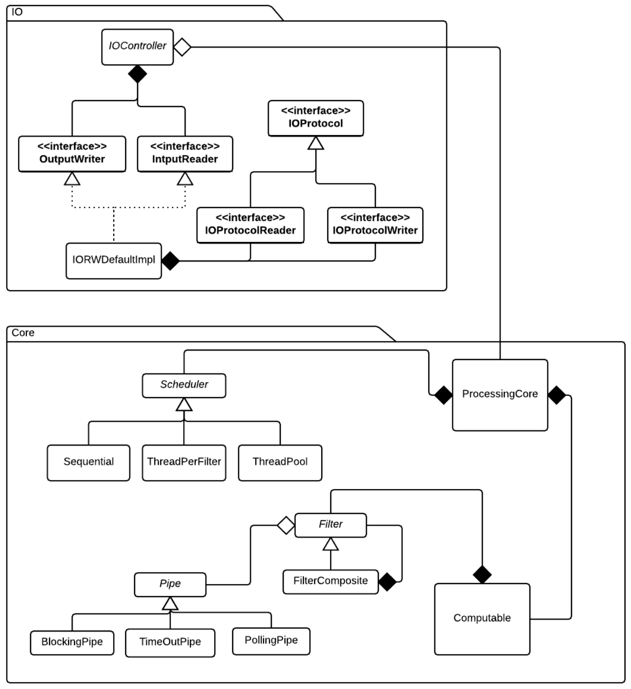

Figure 3: Java Lightweight Processing Framework (JLPF) software architecture

Gaze tracking method
In order to achieve real-time image processing, we have skipped detecting cornea reflection in the image which could compensate for the small movements of the camera relative to the eye. Only the pupil center obtained from the eye image is used for gaze estimation. Pupil detection is done by applying a simple blob detection algorithm on the eye image as follow: (step 1) The image is first converted to grey-scale (step 2) and then a morphological operation is done on the resulting image. (step 3) Then a threshold was applied at a constant level of around 70. (step 4) After thresholding, a morphological operation is done on the image before applying blob detection (4). To reduce the computation time, in each frame, we have defined a region of interest (ROI) for which the image processing is applied for. In the first frame, the ROI will be defined as the entire image. Once the pupil is found on previous processed frames, the ROI is reduced to 30% of the image size and is moved to the most recently computed pupil coordinates (the last frame whose processing is completed).

Calibration
In the current implementation of the system, a homography transformation is used as our gaze mapping function. The mapping function is obtained from a calibration process consisting of a minimum of 4 calibration markers on the display.

GlassGaze: Gaze Tracker Client on Google Glass

GlassGaze [1] is an android app developed for Google Glass that was originally developed to work as a client for the open-source Haytham gaze tracker [1]. GlassGaze provides a convenient user interface that can be controlled by voice and finger gesture. This client has an Android background service that receives the gaze data from the server and allows different applications, to subscribe to its messages. This background service also allows applications on Glass to communicate with the gaze tracking server. By applying the same messaging protocol defined in the GlassGaze we could easily use this app as our client.

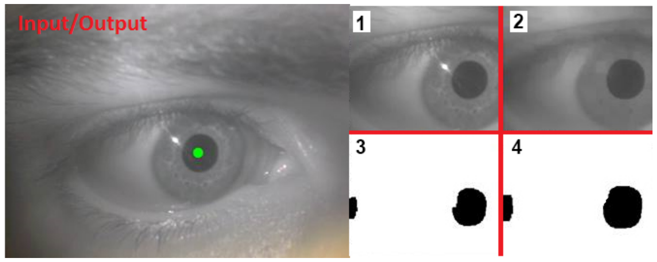

Figure 4: Pupil detection steps

System Performance

Accuracy of the Gaze Tracker

To measure the accuracy of our gaze tracker we conducted an experiment with 10 participants recruited among students from our university. Participants were asked to wear the Google Glass and run the GlassGaze application. First, we had a training session in which they tried the system for a while until they felt comfortable with the system. We started the experiment with a four-point calibration. After calibration, 15 markers were displayed randomly on the Google Glass display for one second. The participants were asked to look at the markers immediately after marker appearance (see Figure 5). We had 15 markers distributed evenly in 3 rows and 5 columns (Figure 6).

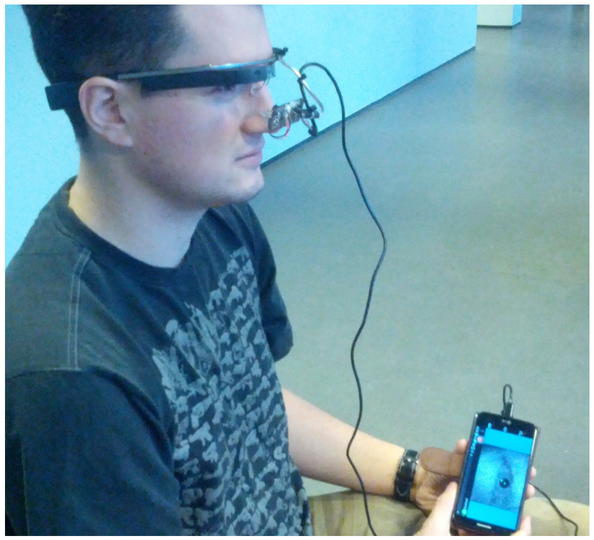

Figure 5: A participant perfomring the task.

To measure the gaze coordinate, the average coordinate of the gaze for the last 700 milliseconds of looking at each marker was calculated. The average of deviation from actual marker position was equal to 52.61 pixels with standard deviation of 35.26 pixels. This means that the error of our gaze tracking system is equal to 1.06 degrees. The distribution of the error for each marker is illustrated in Figure 6. The X and Y dimensions of the graph in Figure 6 represent two dimensions of the display on Google Glass (maximum 640 $\times$ 320), and the gray circles around each marker show the average error of the detected gaze point for each marker.

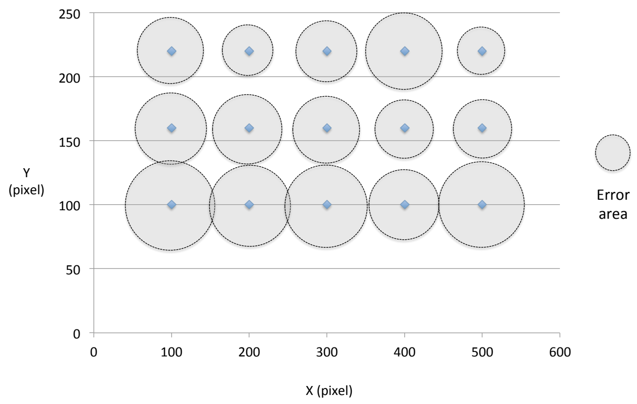

Figure 6: The average error of the gaze estimation for each marker (dots) is illustrated by circles around the markers

Battery Life

To calculate the battery life of the mobile device (a brand new LG-G2 smartphone with 2 GB RAM, a Quad-core 2.26 GHz Krait 400 processor, an Adreno 330 GPU and running Android 4.4 version) while running the EyeDroid application, we measured the charge of the battery (given by the Android built-in battery level indicator) every hour for 3 hours. To compensate the inaccuracy of the built-in indicator, the device was fully charged before conducting each experiment, any other apps were closed but default Android services, and the brightness of the screen was minimized. Each measurement was repeated three times and results were averaged.

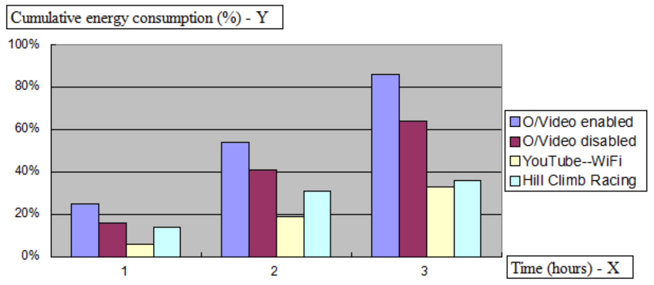

Figure 7: Comparison between EyeDroid and two other popular applications showing cumulative energy consumption (%) per hour

Since EyeDroid can optionally show the resulting coordinates drawn in top of the input video streamed on the device display, two different experiments were conducted. First in the video preview enabled mode and second when the preview mode is disabled. To have a baseline for our comparison, the battery life of the device running two other popular applications was measured in the same way: YouTube video streaming and Hill Climbing racing game. The results suggest that EyeDroid behaves similar to Hill Climbing game but deviating approximately 10% per hour. The maximum battery life estimation running EyeDroid with in preview-disabled-mode is approximately 4.5 hours.

Discussion & Conclusion

In this paper we presented a monocular mobile gaze tracker on Android smartphone to support gaze-based interaction with eyewear devices. We used an efficient and lightweight software architecture to divide image processing task into parallel threads. Using our approach, we reached to 6.41 fps performance in the image processing task. The experimental study showed the accuracy of 1.06 degree for our gaze tracker. The error areas (gray zones) around each marker in the Figure 6 show that our gaze tracker can be used for interaction with Google Glass since users are able to accurately point to (at least) 15 different objects on the display. Although, the head gear was fixed on the head, small movements of the camera relative to the eye could create a relatively large error in the gaze tracking result. This was due to the fact that gaze mapping was using only pupil center. As future work, we will add glint detection to increase robustness of the system.

Acknowledgments

This work was supported by the EU Marie Curie Network iCareNet under grant number 264738.

References

[1] Haytham gaze tracker. http://itu.dk/research/eye/, June 2014.

[2] Ferhat, O., Vilarino, F., and Sánchez, F. A cheap portable eye-tracker solution for common setups. *Journal of Eye Movement Research 7*, 3 (2014), 2.

[3] Jalaliniya, S., Mardanbeigi, D., Pederson, T., and Hansen, D. Head and eye movement as pointing modalities for eyewear computers. In *BSN Workshops, 2014 11th International Conference on* (June 2014), 50–53.

[4] Lab, K. Usage of usb webcam with customized galaxy nexus (android 4.0.3). http://brain.cc.kogakuin.ac.jp/research/usb-e.html.

[5] Li, D., Babcock, J., and Parkhurst, D. J. openeyes: A low-cost head-mounted eye-tracking solution. In *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*, ETRA '06, ACM (New York, NY, USA, 2006), 95–100.

[6] Lukander, K., Jagadeesan, S., Chi, H., and Müller, K. Omg!: A new robust, wearable and affordable open source mobile gaze tracker. In *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services*, MobileHCI '13, ACM (New York, NY, USA, 2013), 408–411.

[7] Mayberry, A., Hu, P., Marlin, B., Salthouse, C., and Ganesan, D. ishadow: Design of a wearable, real-time mobile gaze tracker. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, ACM (New York, NY, USA, 2014), 82–94.

[8] Nel, E., MacKay, D., Zieliński, P., Williams, O., and Cipolla, R. Opengazer: open-source gaze tracker for ordinary webcams.

[9] Rantanen, V., Vanhala, T., Tuisku, O., Niemenlehto, P., Verho, J., Surakka, V., Juhola, M., and Lekkala, J. A wearable, wireless gaze tracker with integrated selection command source for human x2010;computer interaction. *Information Technology in Biomedicine, IEEE Transactions on 15*, 5 (Sept 2011), 795–801.

[10] Starner, T. Project glass: An extension of the self. *Pervasive Computing, IEEE 12*, 2 (April 2013), 14–16.